I proving Software Designs via the Mini u Description Length Principle

Joseph

Gender

Male pronouns have been used in this thesis to refer to people of both sexes in order to smooth the flow of the text rather than imply any sexual bias.

Nomenclature

The word Ada without qualification refers to the Ada83 programming language, defined in Ichbiah et al. (1983).

HOOD, without qualification, is used to refer to HOOD version 3, defined in Delatte et al. (1993). All references to HOOD 4 (HOOD HRM, 1995) are explicit.

A number of words are used in the literature (e.g., function, procedure, operation, and routine) to refer to a similar concept. Frequently, each word has a slightly different meaning; for example, functions are often seen as procedures without side-effects. In this thesis we do not require these distinctions, and so all such words are equivalent. In general we shall use HOOD's term ***operation***.

In this thesis, the word ***object*** is used to refer to a collection of co-operating items, whereas the word ***module*** is generally used to refer to the older concept of sub-progra

Trademark Acknowledgments

A number of trademarks are used in this thesis and for brevity are declared once here as follows but apply throughout the thesis:

Trademark	**Trademark Owner**
Ada	U.S. Department of Defense, Ada Joint Program Office.
ANSI	American National Standards Institute.
AT&T	AT&T.
HOOD	HOOD User Group.
POPLOG	University of Sussex.
PostScript	Adobe, Inc.
SADT	SofTech, Inc.
UNIX	AT&T.

All other trademarks are acknowledged.

Typographic Conventions

A few type definitions are given in Chapter 7, these are presented in VDM, see for example Casey (1994) or Dawes (1991). We have adopted the convention that type-names start with an uppercase letter, and record field names start with a lowercase letter.

List of Figures

List of Tables

Part I

Background

Chapter 1

Introduction

- How is the input of a design to be expressed?
- How are alternative designs created?
- What constitutes a ‘better’ design?

It is the purpose of this thesis to try and answer these questions.

1.2 Motivation

We know from empirical studies (Boehm, 1981), that the cost of correcting defects grows significantly the later in the development process the problem is uncovered. Therefore the more potential errors that are found in the early stages of development reduces the economic costs of owning the software. This potential for significantly decreasing costs means that the design phase of software development is an area which merits further research. Moreover, software design is a sophisticated human skill worthy of study for its insights into other intelligent behaviour.

Most Computer Aided Software Engineering (CASE) tools available today, are little better than glorified drawing packages sometimes with associated databases. Such tools provide support for drawing pictures, and recording information about the software being designed. The more sophisticated systems allow information to be shared by several engineers, and detect improper use of notation and missing elements. Although useful these facilities are limited and perform only a shallow examination of the software being designed. What is needed are tools which sub-modules its ***children***, and the module containing a given sub-module its ***parent***. Modules which may contain sub-modules are called ***nestable***. Modules, unlike humans, can only have ***one parent***. Further, if a module is contained in another (larger) module, then the whole of the sub-module must be contained in the single parent.

When an entity has to be shared between several modules, there is potentially some tension as to which module should 'own' the entity. duTJ169781(i)4.02721(c)5.64.644(p)6.84972(a)5.64311(a)5.64311(u)6.84932

W03T3396(h)6.84932(e)5.64422(l)4.02776(d)6.84932(t)-248.035(b)5(o)6.84932(s)5.43798(h)-234.261(m).026663

•

The beginning of wisdom is found in doubting;
by doubting we come to the question,
and by seeking we may come upon the truth.

Chapter 2

Software Design

Synopsis

This chapter examines the meaning of software design in more detail. We start by asking "What is design?", and looking at the variety of different functions that a design has to perform. In particular we shall see that a design is not purely mechanical but captures the value judgements of those who contribute to the design. We shall then look at various ways for capturing designs and briefly review a broad range of design methods. We shall then examine the established properties of a ***good*** design. We conclude by looking at the meaning of architectural design and the idea of a design as a graph.

2.1 Design Theory

Design[1]

design is implemented, the resultant system will satisfy the requirements (see Figure 2.1). This would also suggest that the success of a design cannot be isolated from its implementation.

importance (see Section 2.7). That is, the correct identification and connectivity of components is essential to the design meeting its requirements; however, simply connecting a set of components at random does not of itself constitute a design, the whole must be a unified system.

Dasgupta also makes the observation that a design form must serve as a user guide. At first this may seem strange to software engineers who are used to separate user guides. Nonetheless, we do expect this information in a design form. Given a new object, the first few questions are likely to be "what does it do, and how do I use it?", i.e., we want a user guide. Only when we have received satisfactory answers to these questions, do we inquire into the connectivity of the object.[8]

Dasgupta's final requirement for a design form is normally not addressed by software design methods, and its absence is responsible for much current research in software and Computer Supported Collaborative Work (CSCW); a little reflection confirms that it is a necessary condition. The design form must capture the justification (and history) of a design, so that it can be critically examined and support changes. That is, the design form must encapsulate some notion of why this is the preferred design. An immediate consequence is to change the nature of the design from a static document to a dynamic form. This area is fraught with difficulties, firstly because of the volume of information and secondly the designer may be reluctant to explain his reasoning due to satisficing (see Section 2.1.5).

These differing requirements for the design form, are captured diagrammatically in Figure 2.2.

CRITI

This language need not be the same as the previous language.

- The target implementation language. This impacts on the types of abstraction which will be considered by the designer. Whilst it is true that all software ultimately runs in machine code, some languages are better suited to specific task

development and expression of a specification. Formal notations are intended to be accompanied by a natural language description of what the mathematics is modelling.

Formal languages have the advantage of being precise, unambiguous and amenable to rigorous analysis using all the leverage that mathematics can bring to bear. Moreover they permit the engineer to move away from the fuzzy languages used in the initial specification, and use a more abstract and precise notation. Precise notation allows the designer to look for missing parts of the design/specification and ambiguities, whilst also permitting a more abstract model to be developed which allows alternatives to be explored.

However, formal languages are not without their problems. Most notably their very reliance on mathematical notation and reasoning which the average engineer is unfamiliar with. This is not unreasonable since the software engineers must communicate with customers and other non-specialists. Also as Jackson (1995, p.116) has noted “formalists often forget the need to tie their descriptions to the reality they describe”. Fetzer (1988) observed that it is impossible to mechanically (completely) derive an implementation from a specification, which some advocates of formal methods seem to believe. The cost (in terms of time) of producing a formal model, can be quite high and may not be justifiable in terms of the benefits to the project.

There are undoubted areas on some projects where the advantages of formal methods outweigh their disadvantages, but they should not be seen as a panacea, but rather as a valuable part of software engineering’s toolkit.

2.3.4 Choice of Language

By this stage the reader may be wondering about our choice of design notation. We believe that pictures

- Modular Approach to Software Construction Operation and Test (MASCOT) (MASCOT, 1987)

Let us briefly consider stepwise refinement as an example of this design methodology. Stepwise refinement was proposed by Wirth (1971). The design is developed by successively refining the previous procedural detail. Thus a system is progressively decomposed from high level functional statements until programming language statements are reached. This process can be though of as elaborating the design, at each iteration we provide more detail.

At least three different “rules” for refinement have been identified, namely (Grogono, 1980): divide and conquer, make finite progress, and analyse cases. It is important to realise that at each iteration a decision (there are always choices) must be made on the “best” way to proceed. Following this method can lead to dead-ends, and therefore it may be necessary to backtrack and re-iterate again.

The method is not prescriptive and does not guarantee a solution, nor indeed does it always provide a notation. It is heavily biased towards the Waterfall model, and is often used as a basis for teaching design.

The criticisms raised against functional decomposition stem from three main observations. Firstly, the top level decomposition must be made when knowledge of the problem is least developed and the method offers no certainty that we have identified the top level function correctly or that our refinement is not a blind alley. (Think of this as a search, are we starting from the root node and which child do we visit next?) Secondly, Jackson (1983) has argued that the functions change over the life of the system as opposed to the structure of the data. Thirdly, the design of key data structures etc. can permeate the entire program.

It is the second and third problems have led to the evolution of object-oriented design.

2.4.2 Data Structured Design

These methods seek to mould the program (structure) to the structure of the data. An archetypal example is file handling. These methods do not attempt to model the flow of data through the system, but rather the static structure of the data. Examples of these methods include:

- Jackson Structured Programming (JSP) (Jackson, 1975)
- Jackson System Development (JSD) (Jackson, 1983)
- Warnier-Orr (Orr, 1971)

The major problem with these methods is their rigidity; the necessity to identity the data’s structure. Additionally implementations tend to be slow; JSD tends to lead to a large number of processes, and context switching is expensive (Deitel, 1984). JSP tends to be more mechanistic than some other design methods, and has been used as the basis for some undergraduate design courses. However JSP can lead to dead ends caused by structure clashes due to discrepancies between different real-world data structures.

2.4.3 Object Oriented Design

In this group of methods, the problem domain is seen as being composed of objects and classes of objects. An object encapsulates both algorithms and data. Objects are potentially related to each other in a variety of ways, not all of which are hierarchical in nature. For example, a filled red

- Vienna Design Method (VDM) (Jones, 1986)
-

6. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.

Pressman (1992, p.318)

Although the term module is used above, it is clear that our definition of object could equally be used in its place.

2.6 Architectural Design

In this section we look in more detail at the concept of architectural design. In particular we examine why we regard architectural design as more significant to the success or failure of a design than detailed design.

By architectural design we mean the identification of the major components of the design, especially their purposes and interfaces. How we can see the reason why we claim that detailed design is less important; detailed design is concerned with designing the internals of the identified components. As Fowler and Scott (1997, p.22) observed " ... the biggest technological risks are inherent in how the components of a design fit together, rather than present in any of the components themselves". Moreover, designing the internals is obviously a much smaller and self-contained problem than the original problem.

In practice, of course, once a 'large' component has been identified, the design of its internal structure is also architectural in nature not just detailed design. We regard the architecture of 'large' components to be part of the architectural design phase. Specifically we classify detailed design as deciding ***how*** a component's services should be provided rather than deciding ***what*** services should be provided.

We saw in the previous section that good design requires objects which are largely independent and have a good logical structure. These two concepts are captured by ***loose coupling*** and ***high cohesion***, respectively. These concepts are further examined below, after we have described exactly what is meant by a component.

2.6.1 What is an Object?

So far, we have been deliberately rather vague about what we mean by a software component or object. We now offer a more precise definition.

> An object is a model of a real-world entity or a software solution entity that combines data and operations in such way that data are encapsulated in the object and are accessed through the operations. An object thus provides operations for other objects, and may in turn also require operations of another object. An object may have a state, either explicitly to provide control or implicitly in terms of the value of the internal data.
>
> Robinson (1992a, p.34)

This definition accords with Pressman (1992) earlier properties for good design, and gives us a good definition of an object. It is important to note that an object (generally) both provides services to other objects and requires services from other objects. This definition does not rule out mutual recursion, but normally this is rare.

Most modern programming provide the object concept, albeit under a variety of different names: class, cluster, module, package and structure.

2.6.2 Are these the Right Objects?

Having defined the term 'object', and a definition of good design properties, we how require some guidance on determining the quality of a proposed architectural design.

books, which results in communicational cohesion and hence is traditionally considered unsatisfactory. However, in OOD using an object to represent an abstract data type is considered good practice.

A generally accepted cohesion scale from highly desirable to accidental is shown below, (Pressman, 1992, p.334):

Functional Cohesion All components of the module contribute to a ***single*** task.

Sequential Cohesion The module's components are used in some fixed order to perform a task; but it lacks a strong sense of single mindedness.

Communicational Cohesion The components are located in the same module because they use the same input or output data rather than having functional cohesion.

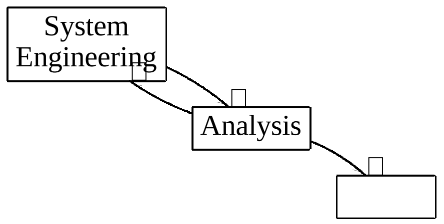
Procedural Cohesion The components are related because they are used in some fixed order at particular moments in time. For example, the use of procedure ***B*** must always be preceded by the use of procedure ***A***.

Unfortunately, this description is a little too simple, because most large designs require some

Chapter 3

An Overview of HOOD

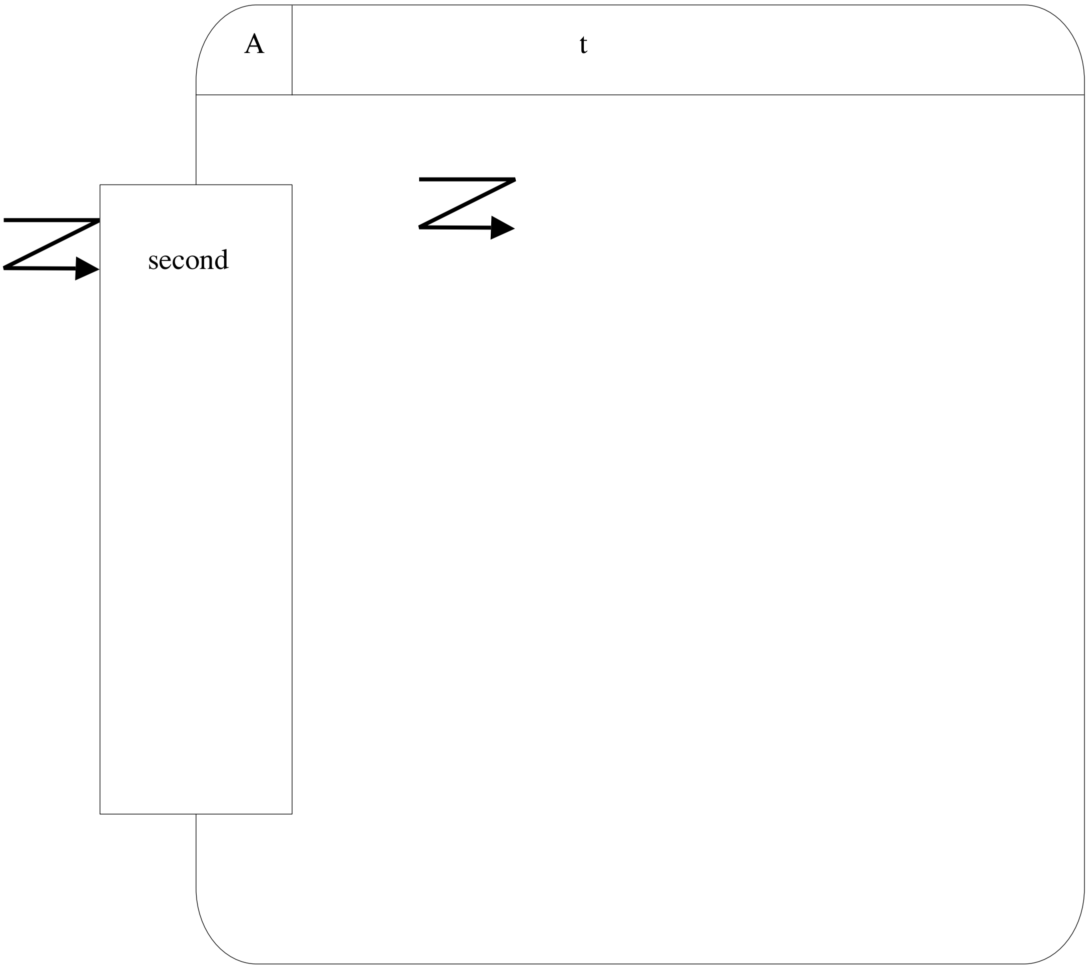
it suffices to use a basic model, called the Waterfall model (Pressman, 1992). The Waterfall model is depicted in Figure 3.1. It must be stressed that the Waterfall is an idealised model, and not a description of what may happen on a real project.



(a) Statement of the problem

3.3 Objects - Architectural Components

HOOD regards ***objects*** as the architectural building blocks. This section explains the nature of objects, and in particular objects in HOOD


A
t
second

terminal object has ***at least two child objects***

3.4.5.2 Inter-Object Visibility

The question of inter-object visibility really boils down to the question, what objects are visible to the Required Interface of the object being considered. Note that all such entity references must be resolved by including the object's name.

- All environmental objects are visible throughout the system.
- The Provided Interface of all of an object's siblings are visible.
- The Required Interface of the object's parent (if not a root object) is visible.
- Nothing else is visible.

3.5 HOOD Entities

This section describes the entities which may make up a HOOD object.

Types in HOOD

```
OBJECT traffic_lights IS ACTIVE

  DESCRIPTION
    --The traffic lights system controls four traffic lights at a crossroads.
       The traffic sensors inform the system of waiting traffic.--

  IMPLEMENTATION_CONSTRAINTS
    --The system is driven by a 1Hz clock--

  PROVIDED_INTERFACE
    OPERATIONS
      second ;

  OBJECT_CONTROL_STRUCTURE
    DESCRIPTION
      --Each second, traffic_lights is activated to look at the traffic
      sensors and to change the lights.--
    CONSTRAINED_OPERATIONS
      second CONSTRAINED_BY ASER_BY_IT --|#1234|-- ;

  REQUIRED_INTERFACE
    NONE

  INTERNALS
    OBJECTS
      seconds ;
      traffic_sensors ;
      lights ;
    TYPES
      road ; --| is (AC, BD) defines road configuration |--
    OPERATIONS
      second IMPLEMENTED_BY seconds.count ;

    OBJECT_CONTROL_STRUCTURE
      IMPLEMENTED_BY seconds ;

END_OBJECT traffic_lights
```

The **description** section introduces a textual comment describing the problem. It may contain anything the designer wishes. All comments in an ODS are bracketed by '`--{`' and '`}--`'. In addition to comments, an ODS may contain ***free text***, bracketed by '`--|`' and '`|--`'. Free text may only occur in specific places in the ODS, and used as a mechanism for passing additional information to other tools, the text has no ***defined*** meaning in HOOD. The **implementation_constraints** section is used in the same way, but is intended to document im

The design for `lights` is shown below

```
OBJECT lights IS PASSIVE

  DESCRIPTION
    --{Object lights is used to set a traffic light pair to a selected colour;
       allowing for proper sequencing of all lights as necessary for safety.}--

  IMPLEMENTATION_CONSTRAINTS
    --{In this simulation, text_io is used to provide a readable output.}--

  PROVIDED_INTERFACE
    TYPES
      colour ; --| is ( RED, RED_AMBER, GREEN, AMBER ) |--
    OPERATIONS
      change ( road_name : IN traffic_lights.road ;
               to_colour : IN colour ) ;

  REQUIRED_INTERFACE
    OBJECT traffic_lights
      TYPES
        road ;

    OBJECT text_io
      TYPES
        string ;
      OPERATIONS
        put_line ( item : IN string ) ;  --| print a string |--

  INTERNALS
    DATA
      other_road : traffic_lights.road ;

    OPERATION_CONTROL_STRUCTURES

      OPERATION change ( road_name : IN traffic_lights.road ;
                         to_colour : IN colour )
        DESCRIPTION
          --{The data item other_road is initialised to the opposite of the
             value of road_name. If the requested colour is GREEN, operation
             change controls the full sequencing from GREEN to AMBER to RED
             for one light set, and RED to RED-AMBER to GREEN for the other
             light set.
             If the requested colour is RED or AMBER, operation change simply
             sets the requested light to RED or AMBER.}--

        USED_OPERATIONS
          text_io.put_line ( item : IN string ) ;

        PSEUDO_CODE
          --|if road_name = AC then
               set other_road = BD
             else
               set other_road = AC
             end if ;
             if to_colour = GREEN then
               set other_road lights to AMBER ;
               set road_name  lights to RED-AMBER :
```

```
              set other_road lights to RED ;
              set road_name  lights to GREEN :
            else
              set road_name lights to to_colour ;
            endif |--
      END_OPERATION change

END_OBJECT lights
```

Much of this is as for `traffic_lights` so we will only discuss the new sections.

The **required_interface** section now says that `lights` requires types `traffic_lights.road` and `text_io.string`, in addition to the operation `text_io.put_line`.

The new section **operation_control_structures** contains an entry for each operation declared in the **internals**. Each operation is described as required. This is followed by a list of used operations, and optionally as comments) **pseudo_code** and the final **code**.

The designs for `seconds` and `traffic_sensors` are shown below

```
OBJECT seconds IS ACTIVE

  DESCRIPTION
    --{Object seconds is activated from its parent object traffic_lights by the
       operation traffic_lights.second. It checks for traffic and changes the
       lights if appropriate.
       Seconds keeps a count of the time since the last light change and the
       road pair that is GREEN (AC/BD).
       After 40/20 seconds elapsed, seconds checks the traffic_sensors each
       second. When the traffic sensors show that there is traffic waiting at
       the other road, the lights are changed.}--

  IMPLEMENTATION_OR_SYNCHRONISATION_CONSTRAINTS
    --{Operation count of object seconds is activated once every second by
       interrupt at address 1234.}--

  PROVIDED_INTERFACE
```

```
             to_colour : IN colour ) ;

OBJECT traffic_sensors
  TYPES
    present
  OPERATIONS
```

```
            is_present : IN OUT present ) ;

  REQUIRED_INTERFACE
    OBJECT traffic_lights
      TYPES
        road ;

  INTERNALS
    TYPES
      latch ;
    DATA
      ac_sensors : latch ;
      bd_sensors : latch ;
    OPERATIONS
      read_sensor ( sensor : IN latch ) RETURN present ;
      check ( road_name  : IN traffic_lights.road ;
              is_present : IN OUT present ) ;

    OPERATION_CONTROL_STRUCTURES

      OPERATION check ( road_name  : IN traffic_lights.road ;
                        is_present : IN OUT present )
        DESCRIPTION
          --{Operation check reads the hardware sensors for the road given in
             the parameter road_name to find out if traffic is present on
             either side, and returns the value is_present set to TRUE or
             FALSE.}--
        USED_OPERATIONS
          read_sensor ( sensor : IN latch ) RETURN present ;
      END_OPERATION check

      OPERATION read_sensor ( sensor : IN latch ) RETURN present
        DESCRIPTION
          --{Operation read_sensor reads a hardware sensor at the given sensor
             latch, and returns the value TRUE or FALSE.}--
      END_OPERATION read_sensor

END_OBJECT traffic_sensors
```

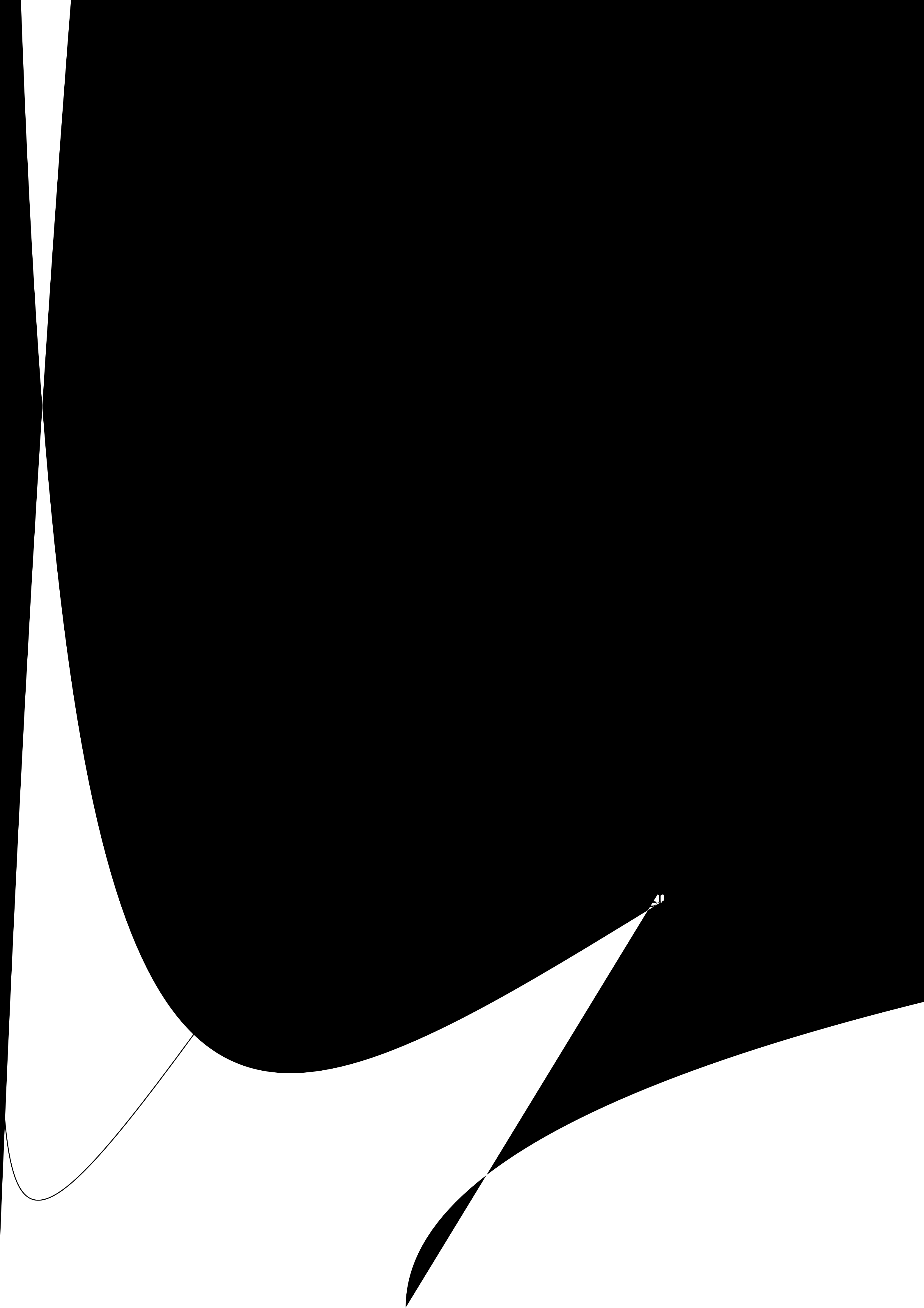
Finally the design of the environmental object `text_io` is below, recall that such objects have nothing except a **provided_interface**.

```
OBJECT text_io IS ENVIRONMENT PASSIVE
  PROVIDED_INTERFACE
    TYPES
      string ;
    OPERATIONS
      put_line ( item : IN string ) ;  -
END_OBJECT text_io
```

3.7 Unused HOOD Facilities

As mentioned in the introduction to this chapter, some features of HOOD were not used in this thesis. This section briefly outlines these unused features, and explains why they were omitted.

Generic Classes in HOOD allow the creation of Ada generic objects, which can then be instantiated to form objects. Their use in HOOD is not very common, and how this kind of information should be handled in our complexity measure is far from clear.



4.1 Requirements for a Complexity Measure

Based on the above overview, we want our complexity measure to satisfy the following requirements

- It must be possible to evaluate a design's complexity without reference to its implementation.
-

- LOC tells us nothing about how to make complex designs less complex.

We have been very critical of LOC, perhaps unfairly as this measure was never meant to reflect design quality or complexity. However, this analysis does serve to lay a framework for discussing other proposed program complexity measures.

The most notable code metrics are Software Science (Halstead, 1977) and Cyclomatic Complexity (McCabe, 1976). Both have been quite well researched; and were initially regarded quite favourably, but more recently their theoretical underpinnings have been shown to be weak (see Shepperd and Ince, 1993, p.28–40).

Since software engineers use such a wide variety of notations, some researchers have tried to extract design information from the resulting program code rather than the design (Shepperd, 1993, p.8), but as Shepperd comments “this must be considered a last resort”. The problem is that the information is available so late and furthermore the code implementation may have an impact on what ***exactly*** is measured.

Clearly, due to their late availability and doubts over their value as complexity metrics, code metrics are unsuitable for our purposes. So we shall now look at some of the proposed design metrics.

4.2.2 Design Metrics

A number of design metrics have been proposed, for example Em

Shepperd reports because distance metrics which yielded intuitive results with one design failed to produce acceptable results for other examples. Even small changes to a design could make the resultant dendrogram unappealing to our intuitive notions of a good design. We conclude therefore that this approach was unsuitable for our purposes.

4.2.3 Object Oriented Design Metrics

As explained earlier this thesis is not based on object oriented design but rather object based design concepts, but given the current interest in the work of Chidamber and Kemerer (1994), we deal briefly with this subject. Chidamber and Kemerer proposed a set of six metrics for measuring a variety of attributes of object-oriented systems (by examining the program code). These attributes are: weighted methods per class, depth of inheritance tree, number of children of a class, coupling between object classes, response for a class (i.e., the number of methods potentially called by a class) and lack of cohesion in methods.

Their work, which has become a ***de facto*** standard for object-oriented metrics, includes a philosophical basis and theoretical validation against the Weyuker (1988) property set for complexity measures. However Chidamber and Kemerer offer no method for trading between the measured attributes, for example coupling and cohesion. Churcher and Shepperd (1995) have also observed that Chidamber and Kemerer definitions need to be made more precise in the light of differences between languages—so that cross comparisons amongst different work can be carried out. Briand et al. (1996) also show that Chidamber and Kemerer metrics do not satisfy their proposed requirements for complexity metrics. However, Chidamber and Kemerer never claim that their metrics were intended to be complexity measures.

4.2.4 Information Theory and Design Metrics

There have been a few measures of software complexity based on information theory. Khoshgoftaar and Allen (1994) survey information theory and software metrics. The following section is derived from their survey findings.

Mohanty (1981) uses a measure of excess entropy[1] to study the information shared between objects. Mohanty regarded this as a measure of interface complexity, but Khoshgoftaar and Allen see this as a measure of object coupling. Whatever Mohanty is measuring, his approach does not offer any form of trade-off between object properties.

Lew et al. (1988) take measurements of several different kinds of connectivity between objects, based on message type (control or data) and the static structure of the exchanged data types, to produce three different entropy measures. These measures are then combined into a single measure of complexity. Lew et al.'s use of distinct measures for different design attributes reflects their different role in a design, but forming a single measure from unrelated sources seems unjustified.

Harrison (1992) proposed a complexity measure based on measuring the entropy of a program in terms of used operations. Harrison's approach is similar in nature to Halstead's Software Sciences and suffers from the obvious problem of being code based rather than design based. However, Harrison did validate his proposed metric against Weyuker's property set, and showed that it should be considered as a contender for measuring complexity. Harrison's metric is quite closely related to our proposed metric (for a given graph), but unfortunately uses out-degree rather than (total-)degree for each node. Furthermore, Harrison does not extend his metric to handle

4.3 Combining Different Measures

We have already observed that design involves trade-offs between different attributes, for example coupling and cohesion. Strictly speaking, however, we cannot make these comparisons, because entities, e.g., procedures, types and variables. There is no reason why both the edges $(\boldsymbol{v},\boldsymbol{w})$ and $(\boldsymbol{w},\boldsymbol{v})$ should not exist in the same graph.

A node $\boldsymbol{v}$ will in general require a set of ***requisitions***, $\mathrm{Req}(\boldsymbol{v})$, and offer a set of ***provisions***, $\mathrm{Prv}(\boldsymbol{v})$, algebraically,

$$\mathrm{Prv}(\boldsymbol{v}) = \bigcup_{\boldsymbol{x}\in\boldsymbol{V}} \mathrm{EC}(\boldsymbol{v},\boldsymbol{x})$$
$$\mathrm{Req}(\boldsymbol{v}) = \bigcup_{\boldsymbol{x}\in\boldsymbol{V}} \mathrm{EC}(\boldsymbol{x},\boldsymbol{v})$$

However, most languages do not offer precise control over imports and exports, so Müller et al. defines ***exact requisitions***, $\mathrm{ER}(\boldsymbol{v},\boldsymbol{w})$, (of $\boldsymbol{v}$ from $\boldsymbol{w}$) and ***exact provisions***, $\mathrm{EP}(\boldsymbol{v},\boldsymbol{w})$, (of $\boldsymbol{v}$ to $\boldsymbol{w}$) between nodes, which can be calculated as below

$$\mathrm{ER}(\boldsymbol{v},\boldsymbol{w}) = \mathrm{Req}(\boldsymbol{v}) \cap \mathrm{Prv}(\boldsymbol{w})$$
$$\mathrm{EP}(\boldsymbol{v},\boldsymbol{w}) = \mathrm{Prv}(\boldsymbol{v}) \cap \mathrm{Req}(\boldsymbol{w})$$

Having defined exact provisions and exact requirements, Müller et al. now defines a measure of ***interconnection strength***, $\mathrm{IS}(\boldsymbol{v},\boldsymbol{w})$, as the exact number of resources flowing between the two nodes $\boldsymbol{v}$ and $\boldsymbol{w}$ as

$$\mathrm{IS}(\boldsymbol{v},\boldsymbol{w}) = |\mathrm{ER}(\boldsymbol{v},\boldsymbol{w})| + |\mathrm{EP}(\boldsymbol{v},\boldsymbol{w})|$$

their designs, we have been unable to carry out any empirical validation. However, we have conducted a number of informal experiments (see Chapter 8) and we are satisfied that the measure is reasonable.

Weyuker (1988) proposed a set of nine properties which any measures of ***program complexity*** should satisfy. Her proposal is widely accepted (Shepperd and Ince, 1993) as a basis for theoretical validation, although it has some shortcomings. For example Fenton (1994) argues that two of Weyuker's properties (i.e., W-Property 6.5 and 6.6, see Chapter 6) capture different notions of complexity versus comprehension. However, we do not see why different members of a set of properties should not try to capture different aspects of a relationship. We would be concerned if the property set was internally inconsistent.

Briand et al. (1996) proposed a set Che fi0cs snrthChsteetecomplexity

Briandet al.nanly renurasedWeyuken'l pr snr

Part II

Theory

Chapter 5

Mathematical Background

Synopsis

This chapter provides the necessary mathematical background for understanding the

The components of a graph are obviously disjoint, and hence form a partition of the graph.

Definition 5.11 (Connected Graph). ***A graph with exactly one component is called a*** connected graph**.**

Definition 5.12 (Trees). ***A*** tree ***is an acyclic graph*** $G(\quad, \mathcal{E})$ ***in which one node*** n_r ***has no predecessors and every other node has exactly one predecessor. The node*** n_r ***is called the*** root ***of the tree. A set of trees is called a*** forest**.**

(a) $G_a(\ _a, \mathcal{E}_a)$ (b) $G_b(\ _b, \mathcal{E}_b)$

Figure 5.3: Two graphs

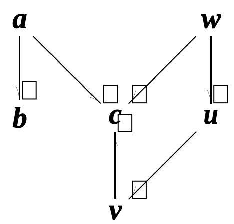


Figure 5.4: Graph union

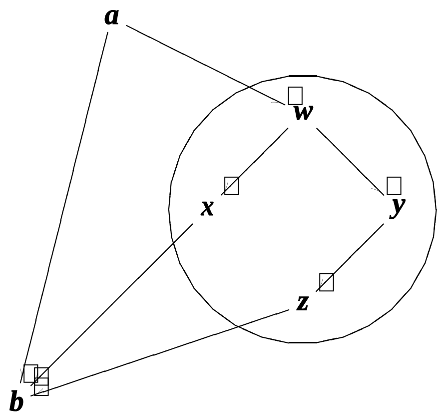
The example in Figure 5.3, has two nodes $\{c, v\}$ and an edge (v, c) in common. If one graph had instead had an edge (c, v), there would have been two edges (v, c) and (c, v) between nodes $\{c, v\}$ in the resulting graph.

If the two graphs being combined have no nodes in common, graph union still yields a single

Such a model is fine for ‘flat’ software architectures, but is not sufficient for true hierarchical designs.

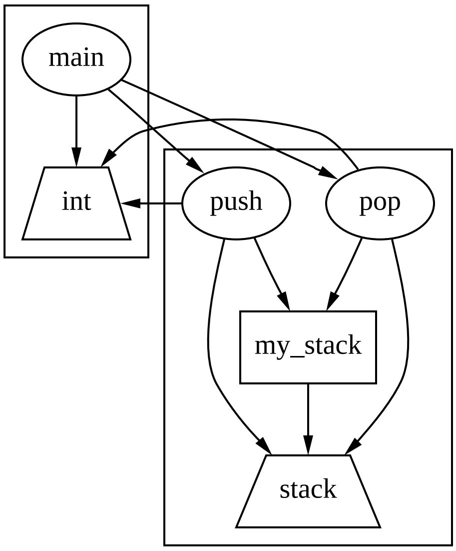
5.1.4 Hierarchical Graphs

The previous sections described standard ‘flat’ graphs, that is every node is just an element of some set. In this section, we introduce the concept of ***hierarchical graphs*** or ***nested graphs***. In a hierarchical graph, a node may itself expand to contain further nodes and edges, and so on ***ad infinitum***.

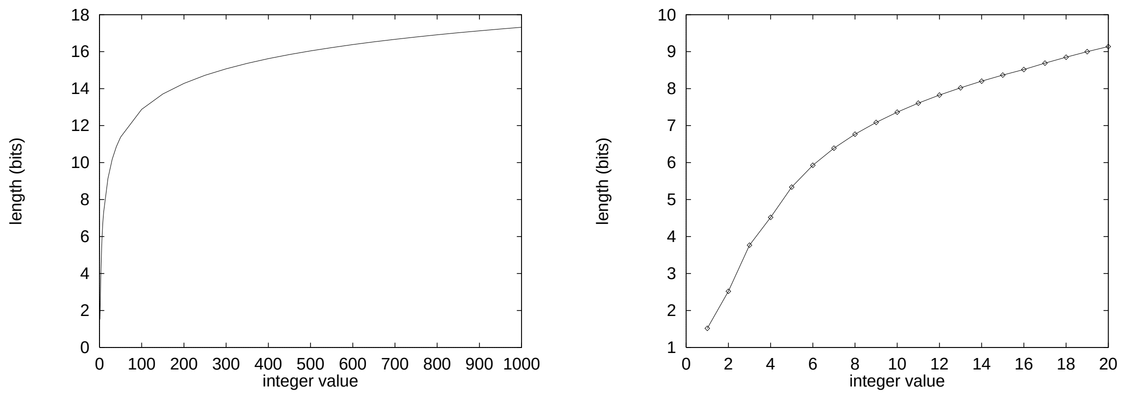


```
OBJECT stack_adt IS PASSIVE
    PROVIDED_INTERFACE
        OPERATIONS
            push ( datum : IN int ) ;
            pop  RETURN int ;
```

```
    pop  RETURN int ;
```


main
int
push
pop
my_stack
stack

To make this clear, consider the transmission of the integer 27_{10}; this has a binary code of 11011_2, and the length of this binary code is clearly 5_{10} which in turn has a binary code of 101_2. Therefore, 27_{10}



(a) Moderate Integers (b) Small Integers

Figure 5.11: Graph of $\log_2^*$

5.2.4 Length of Code Words with Known Probabilities

In Chapter 6, we will see that we need to find the length of a mess two nodes. For our purposes, an isolated node need not be described, since (by implication) it does not contribute anything to the system, otherwise it would be connected to another node.

Hence in our case, we can estimate the minimum message length of a graph's edge description, for the edge $(\boldsymbol{n}_{\boldsymbol{e}_1}, \boldsymbol{n}_{\boldsymbol{e}_2})$ by

$$-\sum_{\boldsymbol{e}\in\mathcal{E}} (\log_2(\Pr \boldsymbol{n}_{\boldsymbol{e}_1}) + \log_2(\Pr \boldsymbol{n}_{\boldsymbol{e}_2}))$$

In our extended model for complexity (see Section 6.2.3) we represent the endpoint of an edge in a hierarchical graph by a sequence of node symbols. (How such a sequence of nodes is determined and how its last element is detected is explained in Chapter 6.) To find the (par-

Chapter 6

Describing a Graph

Synopsis

This chapter presents our complexity measure, Ψ, in detail, explaining how it is calculated and why it is a complexity measure. We also show that Ψ satisfies Weyuker's (1988) proposed property set for complexity measures.

In this chapter we describe our complexity measure, Ψ, in detail, and show that it satisfies Weyuker (1988) proposed property set for complexity measures.

6.1 The Message Passing Metaphor

Our message passing paradigm is very simple. We imagine that we need to communicate the structure of a design graph between a transmitter and a receiver along a ***perfect*** transmission medium. That is, the receiver receives exactly what is transmitted, without any errors, duplication, or data loss. The receiver must be able to recreate an equivalent graph from the received message, plus knowledge of the message's structure.

We hypothesise that the length of the resultant message is a measure of the structural complexity of the proposed design. Further, by application of Occam's razor,[1] a smaller message length indicates a better design.

6.2 Ψ: The Complexity of a Design Graph

In this section we describe our proposed complexity measure. This is a recursive definition, so we start by describing a simplified base case, and building upwards.

6.2.1 Describing the Edges in a Graph

In this section, we are going to develop a partial message for describing the edges in a standard graph. We ignore (for now) issues of how this message might be decoded by the receiver. This is only done to simplify the exposition, a decodable message will be covered in Section 6.2.2.

Given a multi-graph $\boldsymbol{G}^*(\quad ,\mathcal{E}^*)$ with ξ nodes $\{\boldsymbol{n}_1,\ldots,\boldsymbol{n}_\xi\}$. A directed link between $\boldsymbol{n_i}$ and $\boldsymbol{n_j}$ can be represented by the sequence $\boldsymbol{P}(\boldsymbol{i})\ \boldsymbol{P}(\boldsymbol{j})$, where ***P02665(h)-234.232(o)6.84932(1.28759(e)-191.624(d)6.847o6.84***

where $\boldsymbol{d_i}$ is the degree of node $\boldsymbol{n_i}$. The message describing the graph has $2\boldsymbol{E}$ symbols. Therefore, the probability of a node occurring in a message is the node's degree divided by the sum of degree of all nodes.

Hence, we can conclude that the message length of an arbitrary node $\boldsymbol{n_i}$ in a message is:

$$-\log_2\left(\frac{\boldsymbol{d_i}}{\boldsymbol{D}}\right)$$

Where $\boldsymbol{d_i}$ is the degree of node $\boldsymbol{n_i}$, and $\boldsymbol{D}$ is $\sum_{\boldsymbol{i}\in} \boldsymbol{d_i}$. Therefore, the total length of a message describing the structure of a multi-graph is

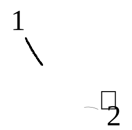
$$-\sum_{\boldsymbol{i}\in} \boldsymbol{d_i}\log_2\left(\frac{\boldsymbol{d_i}}{\boldsymbol{D}}\right) \tag{6.1}$$

We assume that $0 \times \log_2\left(\frac{0}{\boldsymbol{x}}\right) = 0$ for $\boldsymbol{x}$ ■ 0.

Such a (partial) message is sufficient to describe the edges in a multi-graph. This result holds unchanged for a graph, $\boldsymbol{G}(\ ,\mathcal{E})$.

6.2.1.1 Example: Chain Graph

To demonstrate the above, consider the small graph, shown in Figure 6.2. This graph consists of $\boldsymbol{n}$ nodes arranged in a chain, such that, there is an edge from node 1 to node 2, node 2 to node 3, etc. until finally node $\boldsymbol{n} - 1$ has an edge to node $\boldsymbol{n}$, and node $\boldsymbol{n}$ has no other edges impinging on it.



6.2.1.2 Example: Star Graph

Now, consider another small graph, shown in Figure 6.3. This graph consists of $\boldsymbol{n}$ nodes arranged in a star-like configuration, such that, node $\boldsymbol{n}$ has edges to every other node, and the remaining nodes $(1, \ldots, \boldsymbol{n} - 1)$ have no other edge connections dependencies. Note that the same result would be achieved if the direction of every arrow was reversed.

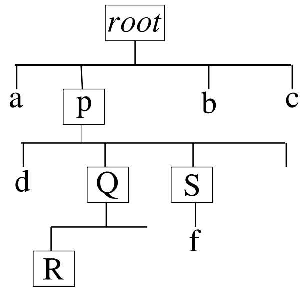
$\boldsymbol{n}$

$1 \quad 2 \quad \cdots \quad \boldsymbol{n} - 2 \quad \boldsymbol{n} - 1$

to add one to the natural number being transmitted so that zero can be sent. Note that this function is strictly greater than zero, for all positive integers, and is strictly monotonically increasing.[5]

Hence the length of a message describing a (multi-)graph is given by

$$\log_2^*(\boldsymbol{E}+1) - \sum_{\boldsymbol{i}\in} \boldsymbol{d_i} \log_2 \left(\frac{\boldsymbol{d_i}}{} \right.$$


root
a
p
b
c
d
Q
S
f
R

is an environmental object. Environmental objects can only be referenced from an object in the current design tree, an environmental object can never reference an object in the design tree. This modification should not distort message length comparisons as it uniformly increases all edges by one bit, and we never change the number of edges in a design graph.

Hence our final model looks like:

Design Description Message

object_description = object_id +
|nodes| +
|objects| {object_description$_1$, ... , object_description$_{\boldsymbol{m}}$} +
|edges| {edge$_1$, ... , edge$_{\boldsymbol{e}}$}

edge = point$_1$ – env_object point$_2$

point = {object_id}* basic_entity_id

env_object = 1 iff next object is an environmental object, 0 otherwise

Length of Full Design Description Message

The length of a message describing a general design using this encoding has the form:

$$\text{length} = \sum_{\boldsymbol{m}\in} \left(\log_2^*(\boldsymbol{N_m}+1) + \log_2^*(\boldsymbol{N'_m}+1) + \log_2^*(\boldsymbol{E_m}+1) - \sum_{\boldsymbol{n}\in\ {}_{\boldsymbol{m}}} \boldsymbol{f_n} \log_2 \left(\frac{\boldsymbol{f_n}}{\boldsymbol{F_n}} \right) + \boldsymbol{E_m} \right) + 1 \tag{6.3}$$

Where is the set of all modules (objects), $\boldsymbol{N_m}$ is the number of entities (objects + basic entities) in module $\boldsymbol{m}$, $\boldsymbol{N'_m}$ is the number of objects in module $\boldsymbol{m}$ ($\boldsymbol{N_m} \geq \boldsymbol{N'_m}$), $\boldsymbol{E_m}$ is the number of edges described in module $\boldsymbol{m}$ and ${}_{\boldsymbol{m}}$ is the set of entities in module $\boldsymbol{m}$. $\boldsymbol{f_n}$ is the frequency of entity $\boldsymbol{n}$ in module $\boldsymbol{m}$, which for basic entities is given by the degree of the corresponding node, and for objects is the degree of the contracted node plus 1 and $\boldsymbol{F_n} = \sum_{\boldsymbol{n}} \boldsymbol{f_n}$.

Length of Single Object Design Description Message

Hence for a design consisting of a single object, we have the following equation for its message length

$$\text{length} = \log_2^*(\boldsymbol{N}+1) + \log_2^*(\boldsymbol{E}+1) - \sum_{\boldsymbol{i}\in} \boldsymbol{d_i} \log_2 \left(\frac{\boldsymbol{d_i}}{2\boldsymbol{E}} \right) + \boldsymbol{E} + \boldsymbol{C} \tag{6.4}$$

Where $\boldsymbol{C} = 1 + \log_2^* 1$, $\boldsymbol{E}$ is the number of edges, $\boldsymbol{N}$ is the number of nodes in the graph, $\boldsymbol{d_i}$ is the degree of the node $\boldsymbol{i}$.

6.3 A Complexity Measure?

Having defined a complexity measure, we should demonstrate, at least informally, that it captures notions of coupling and cohesion. Further that the complexity measure permits some form of trade-off between these two concepts. We will give some more complete examples of this in Chapter 8. For now we will just use an intuitive model.

Consider a design consisting of 10 basic entities, and three objects arranged in a balanced binary tree. Further, let the 10 basic entities form two highly-cohesive groups, with no (or very little) coupling between the groups. Intuitively, it seems reasonable that each group should be placed in a leaf node of the binary tree.

Now consider moving one basic entity from one group, ***L*** into the other ***R***. The `object_description` of the group ***L*** will get smaller, it contains fewer entities and fewer links. The `object_description` of the group ***R*** will get larger, it now contains more entities. Additionally, the `object_description` of the root-group ***T*** will get larger, it contains several links from ***L*** to ***R***. The original design has a complexity of 155 bits, whilst the second design has a complexity of 166 bits. Forming a single object results in a complexity of 172 bits.

However, if the design only contains 6 basic entities, the design has a complexity of 58 bits, but moving all the entities into a single object has a complexity of 52 bits.

and

$$\hat{l} = \log_2^*(\hat{N}+1) + \log_2^*(N^{'}+1) + \log_2^*(E+1) - \sum_{n\in} f_n \log_2\left(\frac{f_n}{F}\right) + E$$

The node is unconnected, so that all the terms in the above sum are unchanged, except $\log_2^*(\hat{N}+1)$. Since $\log_2^* x$ is a strictly monotonically increasing function the length of the module's description must increase. □

Theorem 6.4. ***Adding an additional object to a design graph, increases the design's complexity.***

Proof. Follows immediately from Theorem 6.3 since the node count in the encapsulating object rises and the object count in the encapsulating object rises. □

Theorem 6.5. ***Adding an additional node with degree at least 1 to a design graph, increases the design's complexity.***

Proof. Obvious from proofs of Theorems 6.2 and 6.3. □

W-Property 6.1. ***The measure must not assign the same number to all systems:***

$$\exists p, q \in \quad \bullet \Psi(p) \neq \Psi(q)$$

Proof. Immediately follows from Theorems 6.2–6.5. □

W-Property 6.2. ***There exist only a countable number of systems for a given measurement value.***

> The stated purpose of this axiom is to 'strengthen' the [previous] axiom, as violation suggests that the measure is comparatively insensitive.
>
> Shepperd and Ince (1993, p.68)

Proof. A graph, $G(\quad, \mathcal{E})$, consists of two countable sets, namely: nodes and edges. It follows immediately that the number of graphs is countable since we have only countable unions of countable sets. □

W-Property 6.3. ***There are systems drawn from the same equivalence class:***

$$\exists p, q \in \quad \bullet \Psi(p) = \Psi(q)$$

Proof. Our proof is by constructing two system with the same measure. Let p be an arbitrary system with an underlying graph such that all nodes do not have identical degree. Let q have exactly the same graph, but with the direction of each edge reversed. Since our complexity measure

W-Property 6.5. ***The measure must be monotonic, wrt. adding components:***

$$\boldsymbol{p},\boldsymbol{q} \in \quad \bullet \Psi(\boldsymbol{p}) \leq \Psi(\boldsymbol{p} \circ \boldsymbol{q}) \ \wedge \ \Psi(\boldsymbol{q}) \leq \Psi(\boldsymbol{p} \circ \boldsymbol{q})$$

Where $\circ$ denotes the concatenation operation (see Section 5.1.6).

Proof. Our proof is by induction on the structure of the graph. Recall that design concatenation is derived from graph union, which is in turn derived from set union. Therefore $\boldsymbol{p} \circ \boldsymbol{q}$ has at least as many nodes as $\max(|\ \ _{\boldsymbol{p}}|, |\ \ _{\boldsymbol{q}}|)$ and has at least as many edges as $\max(|\mathcal{E}_{\boldsymbol{p}}|, |\mathcal{E}_{\boldsymbol{q}}|)$. Hence by structural induction using Theorems 6.2–6.5, the design's complexity cannot decrease, as required. □

This result holds, even if there are no links between the constituent designs $\boldsymbol{p}$ and $\boldsymbol{q}$.

Corollary 6.3. ***The complexity of a design concatenated with itself has the same complexity as the original:***

$$\boldsymbol{p} \in \quad \bullet \Psi(\boldsymbol{p}) = \Psi(\boldsymbol{p} \circ \boldsymbol{p})$$

Proof. Immediately follows from the definition of design concatenation (see Section 5.1.6), since for all sets $\boldsymbol{X}$, $\boldsymbol{X} \cup \boldsymbol{X} = \boldsymbol{X}$. □

W-Property 6.6. ***Concatenation of a system r to another system must not always yield a constant increment to the total complexity measure:***

$$\exists \boldsymbol{p},\boldsymbol{q},\boldsymbol{r} \in \quad \bullet \Psi(\boldsymbol{p}) = \Psi(\boldsymbol{q}) \ \wedge \ \Psi(\boldsymbol{r} \circ \boldsymbol{p}) \neq \Psi(\boldsymbol{r} \circ \boldsymbol{q})$$

Also:

$$\exists \boldsymbol{p},\boldsymbol{q},\boldsymbol{r} \in \quad \bullet \Psi(\boldsymbol{p}) = \Psi(\boldsymbol{q}) \ \wedge \ \Psi(\boldsymbol{p} \circ \boldsymbol{r}) \neq \Psi(\boldsymbol{q} \circ \boldsymbol{r})$$

Proof. Since design concatenation is a commutative operation, we have $\boldsymbol{p} \circ \boldsymbol{q} = \boldsymbol{q} \circ \boldsymbol{p}$ for all $\boldsymbol{p}$ and $\boldsymbol{q}$. From Corollary 6.3 concatenating a design with itself yields a design of the same complexity. Hence let $\boldsymbol{p}$ and $\boldsymbol{q}$ be design graphs as in the proof of W-Property 6.3, and let $\boldsymbol{r} = \boldsymbol{p}$, then $\Psi(\boldsymbol{p}) = \Psi(\boldsymbol{q})$ and $\Psi(\boldsymbol{p} \circ \boldsymbol{r}) = \Psi(\boldsymbol{p})$ but $\Psi(\boldsymbol{q} \circ \boldsymbol{r}) \geq \Psi(\boldsymbol{p} \circ \boldsymbol{r})$ since $\boldsymbol{q} \circ \boldsymbol{r}$ contains more edges than $\boldsymbol{p}$ and by Theorem 6.2 this increase the design's complexity, as required. □

Actually all that is required is that $\boldsymbol{p}$ and $\boldsymbol{r}$ have nodes/edges in common, whilst $\boldsymbol{q}$ has nothing in common with $\boldsymbol{p}$.

W-Property 6.7. ***The measure must be sensitive to the ordering of the system components. Let*** ρ ***be a permutation function, then:***

$$\exists \boldsymbol{p} \in \quad \bullet \Psi(\boldsymbol{p}) \neq \Psi(\rho(\boldsymbol{p}))$$

We interpret ordering to refer to moving entities around the hierarchical structure. For example creating a new object or moving a basic entity from one object to another. Weyuker, was discussing moving program fragments, and we regard moving entities as similar for designs.

Proof. See Section 6.3 for an example. □

W-Property 6.8. ***The measure must be insensitive to renaming changes of system components. Let*** τ ***be a renaming function, then:***

$$\boldsymbol{p} \in \quad \bullet \Psi(\boldsymbol{p}) = \Psi(\tau(\boldsymbol{p}$$

W-Property 6.9. ***Module monotonicity***

$$\exists \boldsymbol{p},\boldsymbol{q} \in \quad \bullet\, \Psi(\boldsymbol{p}) + \Psi(\boldsymbol{q}) < \Psi(\boldsymbol{p} \circ \boldsymbol{q})$$

Part III

orp us

Chapter 7

orp us: A Prototype System

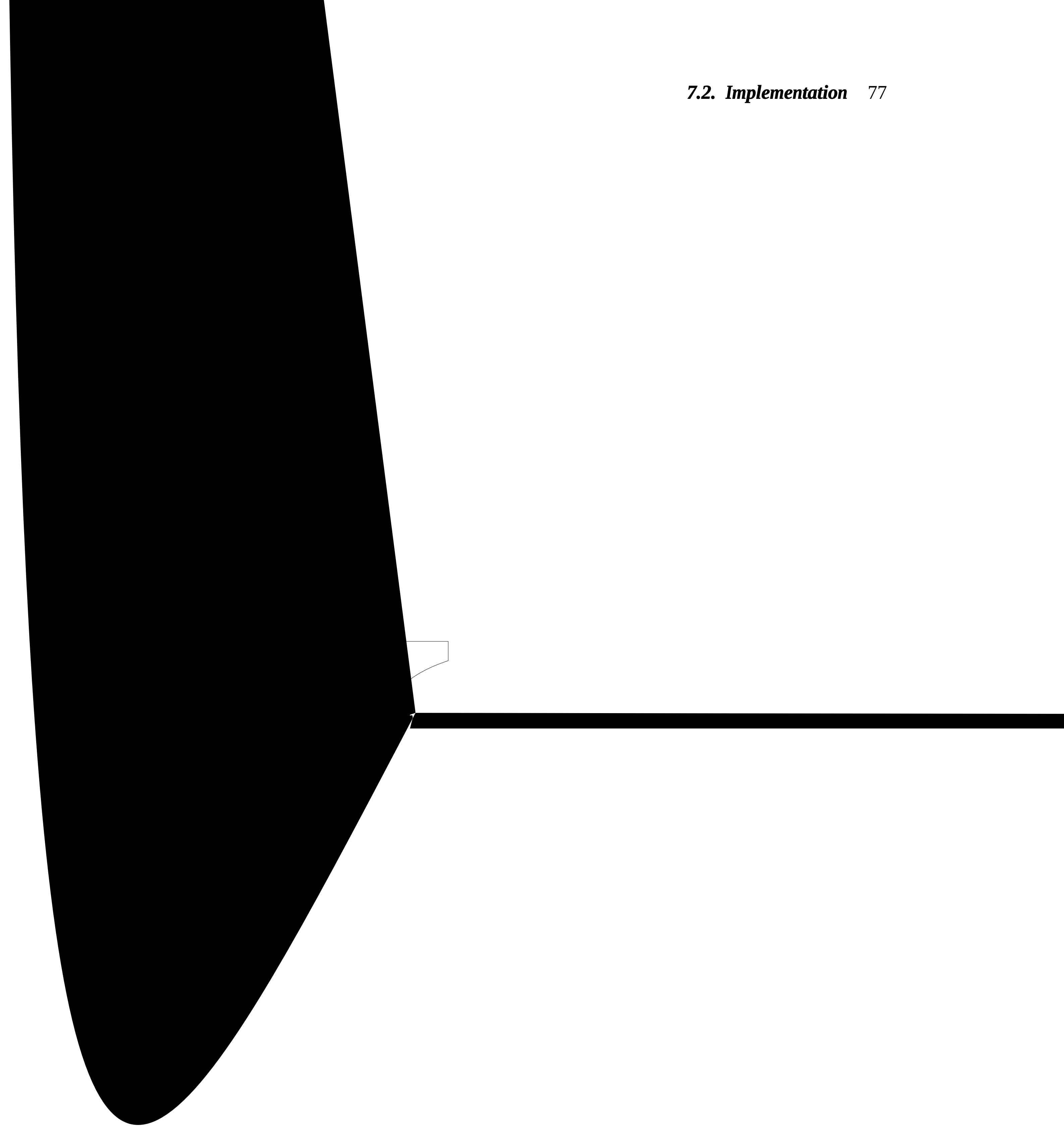
Synopsis

This chapter describes both our extensions to HOOD for capturing a more detailed description of the proposed software architecture and the implementation of our prototype system, *orp us*, for improving designs. *orp us* compares designs based on

code_linkage section. This new section contains information on the types and variables[1] used by the operation. The formal changes to the ODS's syntax are documented in Appendix A.

It was also necessary to make an extension to HOOD's semantics. We permitted the identification of used operations in an operation's definition to include constants as well as operations. This could have been done by adding a further field to the existing ODS for operations. However, since from a general semantic perspective there is little difference between a constant and a procedure, this seems a reasonable change. Additionally, not distinguishing these categories makes mechanical collection easier; a point we shall return to shortly.

It may be objected that these changes impose more housekeeping on the designer and an extra workload for *orp us*



choose to assume, for ease of construction, that the input was essentially error free. If a syntax error is discovered an error message is output and *orp us* halts. Unfortunately,

```
data_analysis ( parse_tree )
begin

    walk parse_tree constructing
```

but a child object does not identify its parent. Fortunately, HOOD requires all object names to be unique within a design tree. This means that as objects are seen, we can construct a table showing the children (if any) of each object. Figure 7.4 shows the structure of the object structure table. On completion of walking the parse tree, we can therefore identify the parent (if one exists) of each object.

Object-Structure = ***Object-Structure-Entry*-set**

Object-Structure-Entry	::	***object-name***	:	***Object-Name***
		parent	:	***Object-Name***
		children	:	***Object-Name*-set**
		siblings	:	***Object-Name*-set**

Figure 7.4: Object Structure Table

This leaves us with two problems. Firstly, we may not have an object tree but rather an object forest, and secondly, we do not know the identity of the root object. Both of these problems can be overcome by creating a pseudo-object (called `$top_object$`) and making its children, those objects which do not have parents.

Entity-Tree = ***Entity-Tree-Node****
Entity-Tree-Node = ***Full-Name*** | ***Entity-Tree***

Figure 7.5: Entity Tree

The entity tree (see Figure 7.5) can now be constructed from the object structure table. The basic-entities can be inserted into the entity tree by scanning the symbol table, and placing each basic-entity declared in a particular object into the corresponding place in the entity tree.

In the supplied design each object has a user-specified name. In principle it is easy to pass this information into the Improvement Engine. However, the activity of the Improvement Engine will create new objects and destroy some existing objects and move entities between objects. Thus rendering the original object name misleading. It was therefore decided to ignore the supplied object name.

7.2.3.4 Deriving the Linkage Information

The second major component of the graph for the Improvement Engine is the set of links. For each entity these links show all the other entities upon which this entity directly depends. As the parse tree is being walked an entity structure record is created for each entity as it is encountered. Figure 7.2.3.4 shows the structure of the entity structure table.

Entity-Structure-Table = ***Entity-Details*-set**

Entity-Details	::	***full-name***	:	***Full-Name***
		kind	:	***Entity-Kind***
		provides	:	***Entities***
		requires	:	***Entities***
		components	:	***Entities***

Entities = ***Full-Name*-set**

Figure 7.6: Entity Structure Table

At the very least each entity provides its own services. The `components` field is only really used by objects and operation_sets, since they are the only encapsulation entities in HOOD. The `requires` field identifies only those other entities directly required by the current entity.

$$\begin{aligned}
\textbf{\textit{Linkage-Table}} &= \textbf{\textit{Entity}} \xrightarrow{m} \textbf{\textit{Depends-On}} \\[1em]
\textbf{\textit{Entity}} &= \textbf{\textit{Full-Name}} \\
\textbf{\textit{Depends-On}} &= \textbf{\textit{Full-Name}}\textbf{-set}
\end{aligned}$$

Figure 7.7: Linkage Table

Once no more changes to the set of entity details is required, construction of the graph's linkage information (see Figure 7.7) is easy. Just use the `requires` field of each entity detail record. No data is generated for entities that have no dependencies, since its node has no outward edges in the underlying graph.

7.2.3.5 Secondary Information

As we noted earlier, there are two minor information requirements due to the nature of HOOD and the Improvement Engine.

$$\begin{aligned}
\textbf{\textit{Environmental-Objects}} &= \textbf{\textit{Objects}} \\
\textbf{\textit{Variables}} &= \textbf{\textit{Entities}} \\[1em]
\textbf{\textit{Objects}} &= \textbf{\textit{Object-Name}}\textbf{-set} \\
\textbf{\textit{Entities}} &= \textbf{\textit{Full-Name}}\textbf{-set}
\end{aligned}$$

Figure 7.8: Secondary Information

A HOOD design must be closed and part of the design philosophy of HOOD is to permit the separate development of individual design components by independent designers (see HOOD HUM (1996)). HOOD

$$\textbf{\textit{History-List}} = \textbf{\textit{History-List-Entries}}^*$$

$$\begin{aligned}\textbf{\textit{History-List-Entries}} :: \ & \textbf{\textit{expanded}} &&: \mathbb{B} \\ & \textbf{\textit{active-entry}} &&: \textbf{\textit{Active-List-Entry}}\end{aligned}$$

Figure 7.12: History List

of the best seen designs is kept. Newly generated designs are compared to the history list, and

7.3.2 Missing Information

As we have noted before *orp us* cannot handle partial designs or designs with missing information. This problem has implications for the deployment of *orp us* in the early stages of design, when its suggestions might be most useful.

It is unworthy of excellent men to lose hours like slaves in the labour of calculation which could be relegated to anyone else if machines were used.

GOTTFRIED WILHELM VON LEIBNITZ (1646–1716)
German philosopher and mathematician

number of nodes	Ψ under null	final	object structure	see

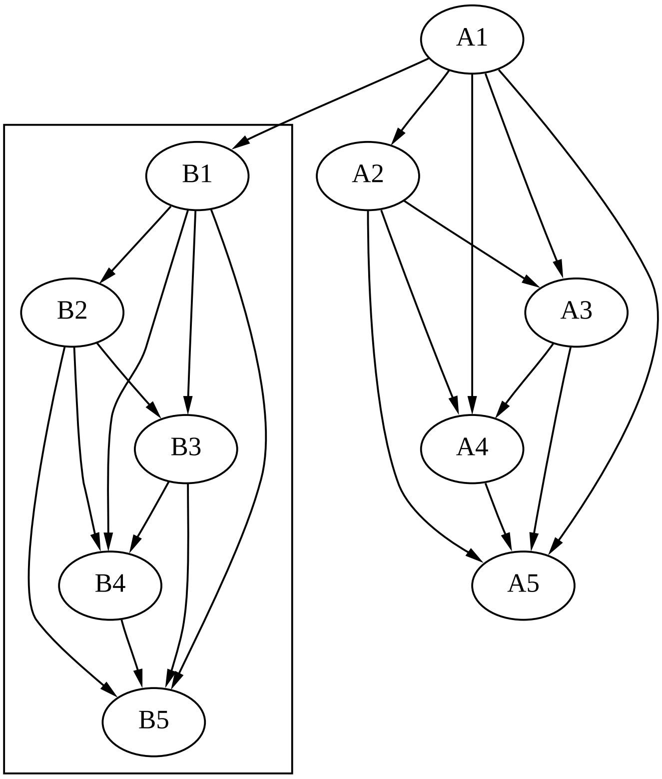


Figure 8.2: Grouping with 2 groups of 5 entities

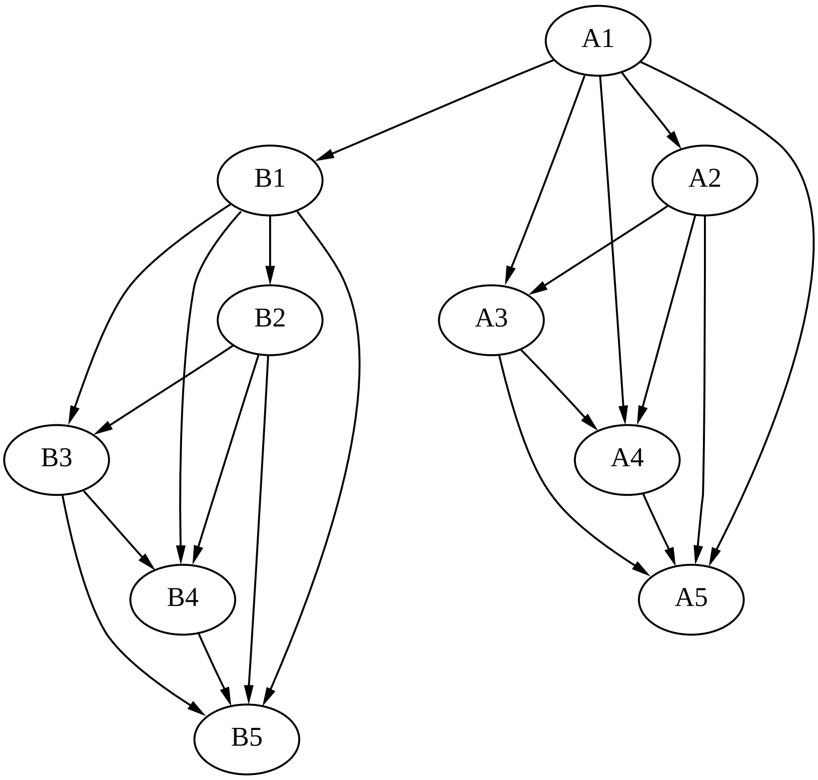
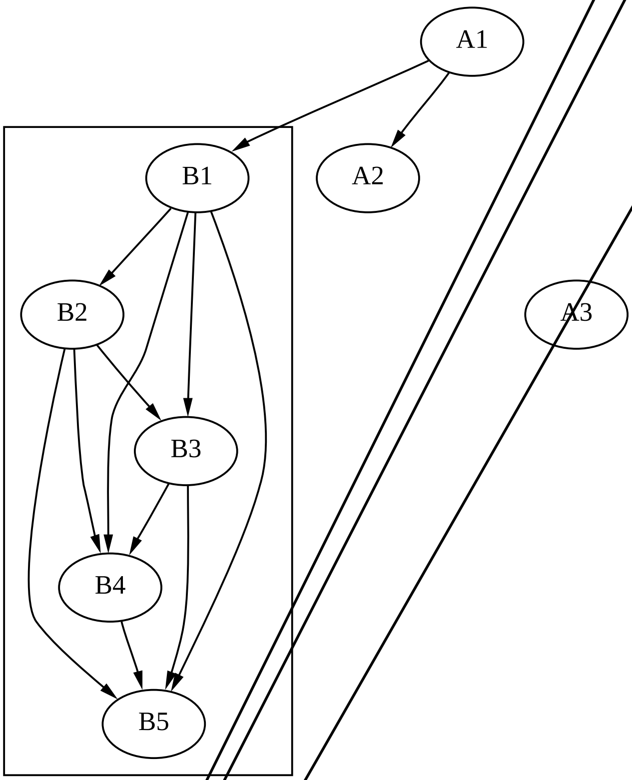


Figure 8.4: No structure with 2 groups of 5 entities


A1
B1
A2
B2
A3
B3
B4
B5

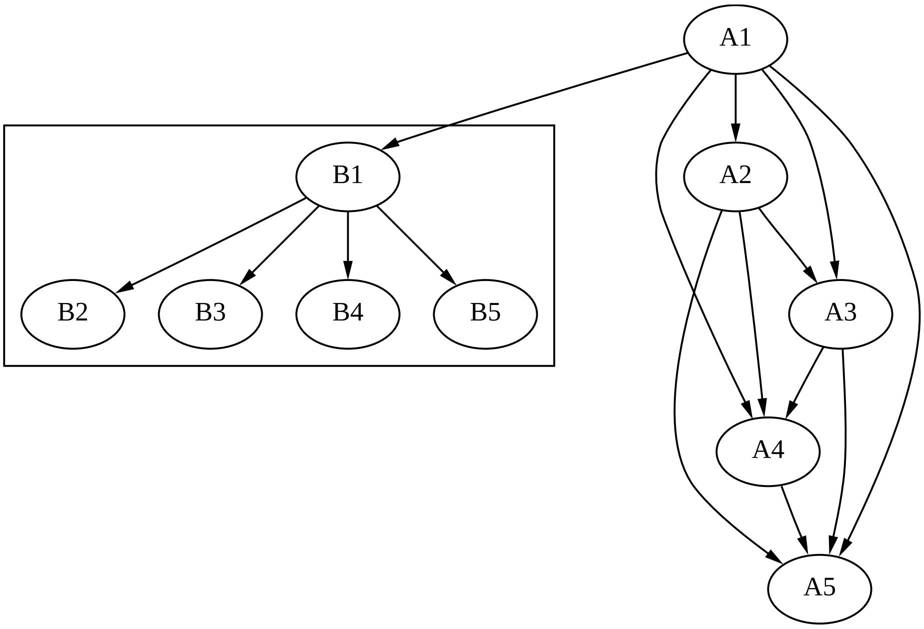


Figure 8.11: Cohesion with 2 groups, one with 4 links

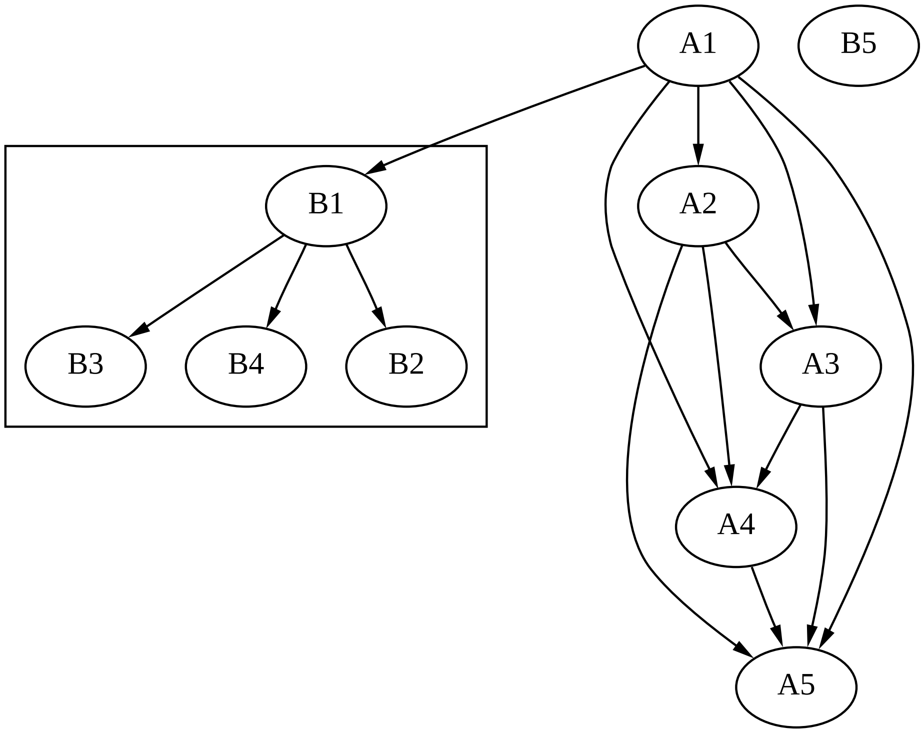
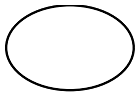


Figure 8.12: Cohesion with 2 groups, one with 3 links



8.1.4 Increasing Coupling

Once again starting from the same base as the previous experiment, we wanted to study the effect of increasing inter-object coupling. We did this by progressively adding links between the ***A*** and ***B*** groups.

We did not want all the new links to go from one basic entity to another basic entity or this would have resulted in a system with a few highly cohesive basic entities rather than just increasing the links from ***A*** to ***B***.

The results are shown in Table 8.3. The double line under the entry starting 5, is a reminder that the linkage structure underwent a change. Between 0 and 5 (inclusive), we formed links from group ***A*** to ***B***, by just adding a link from $\boldsymbol{A_i}$ to $\boldsymbol{B_i}$. However, after 5 we formed additional links by adding links from $\boldsymbol{B_i}$ to $\boldsymbol{A_i}$, the intention being to avoid just making ***B*** a sub-group of ***A***. In retrospect, this idea was correct in principle, but we would have been better to add links from $\boldsymbol{B_i}$ to $\boldsymbol{A}_{6-\boldsymbol{i}}$ thus avoiding too much cohesion between specific basic entities.

number of links	Ψ under null hypothesis	final Ψ	object structure	see figure
0	172.3	157.2	((B1,B2,B3,B4,B5) A1,A2,A3,A4,A5)	8.16
1	179.8	160.8	as above	8.17
2	187.4	170.0	as above	8.18
3	195.2	178.9	as above	8.19
4	203.1	187.8	as above	8.20
5	211.0	196.5	as above	8.21
6	218.8	205.8	((A1,A2,A3,A4,A5) (B1,B2,B3,B4,B5))	8.22
7	226.5	213.8	as above	8.23
8	234.	225.9	(((A3,A4,A5,B3,B4,B5) A2,B2) A1,B1)	8.24
9	242.0	232.3	as above	8.25
10	249.7	238.7	as above	8.26

Table 8.3: Effect of Increasing Coupling between Groups

Looking at Table 8.3, we see that complexity rises as new inter-group links are added, as expected. More interestingly, at 6 links, the two groups are formed into two ‘equal’ sub-objects, which we speculate is caused by trade-offs on the complexity of adding a new object containing 6/7 edges versus the additional complexity of more links between the an encapsulated object and

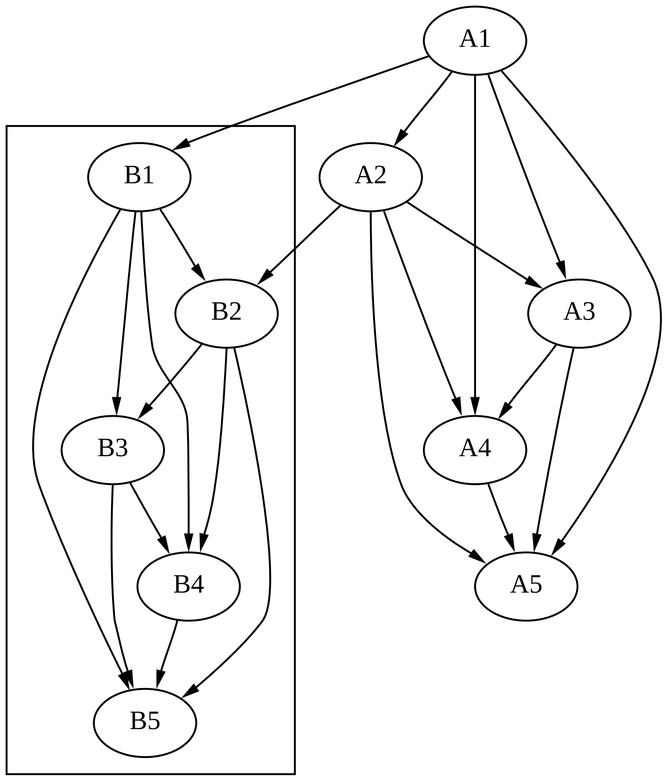
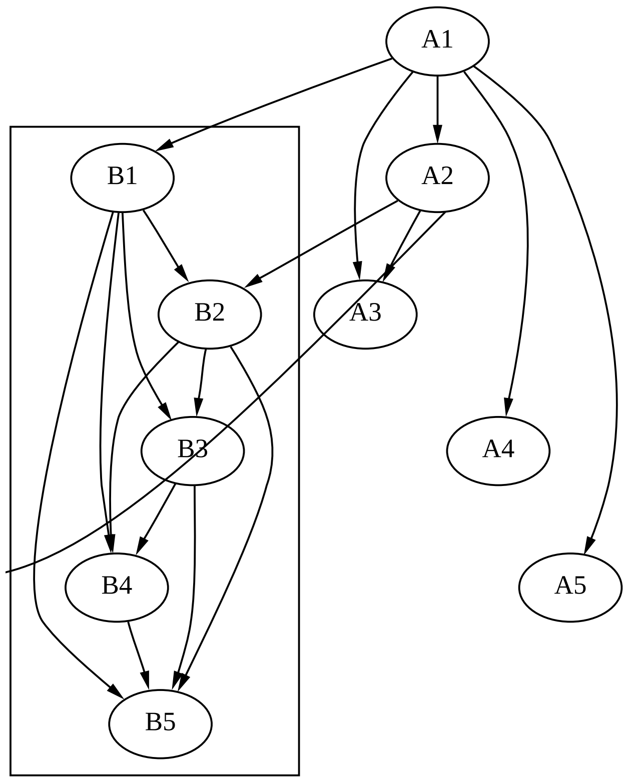
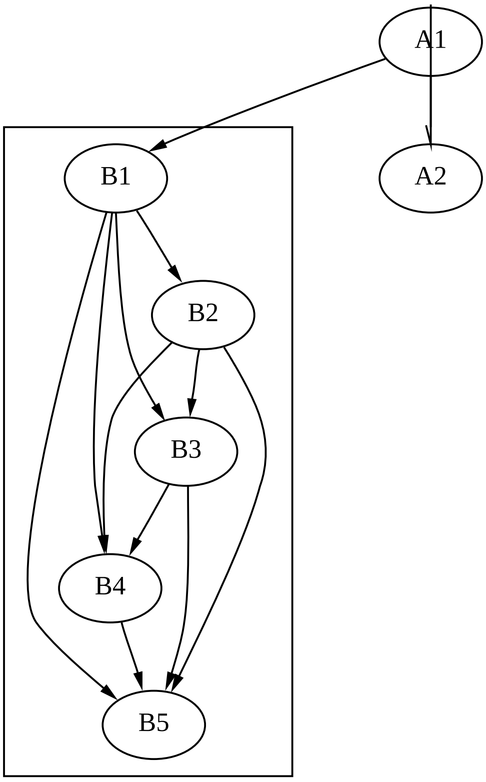
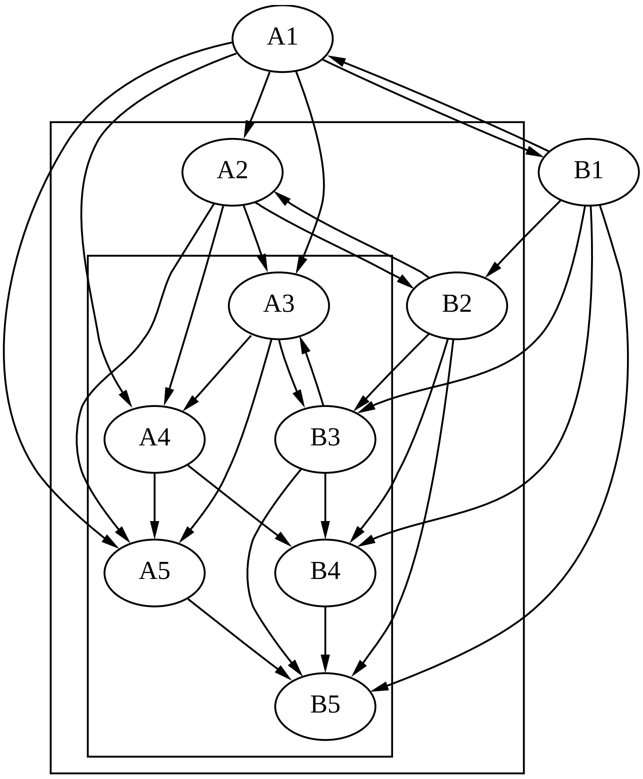


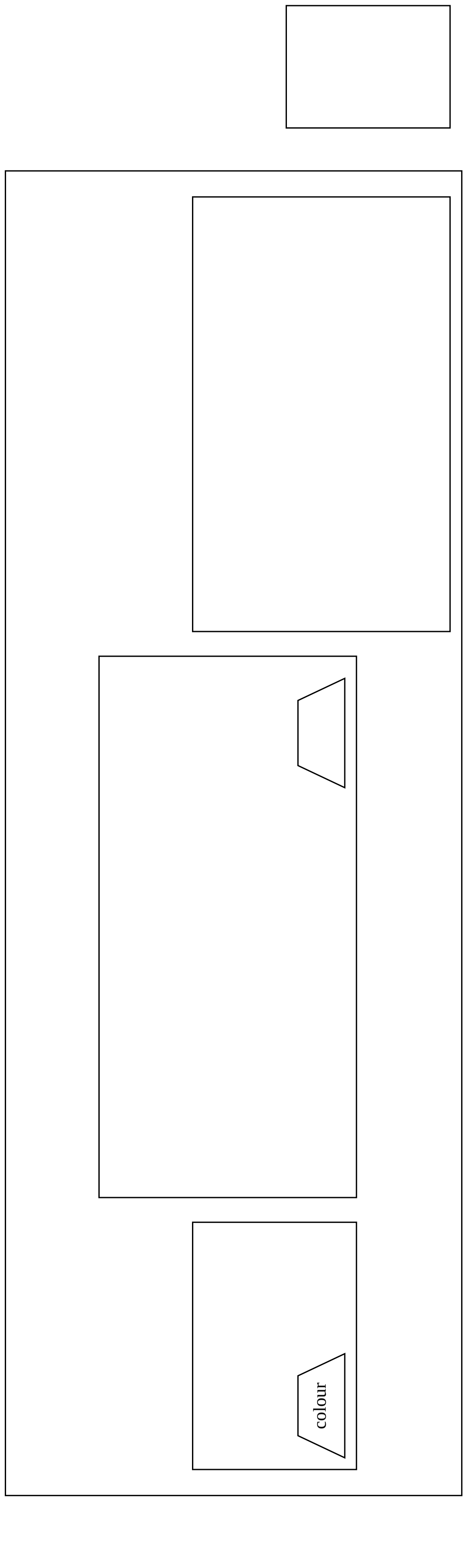
Figure 8.18: Coupling with 2 groups, and 2 links between groups




A1
A2
B1
B2
B3
B4
B5


A1
A2
B1
A3
B2
A4
B3
A5
B4
B5

8.2 A Small Example: Traffic Lights


colour

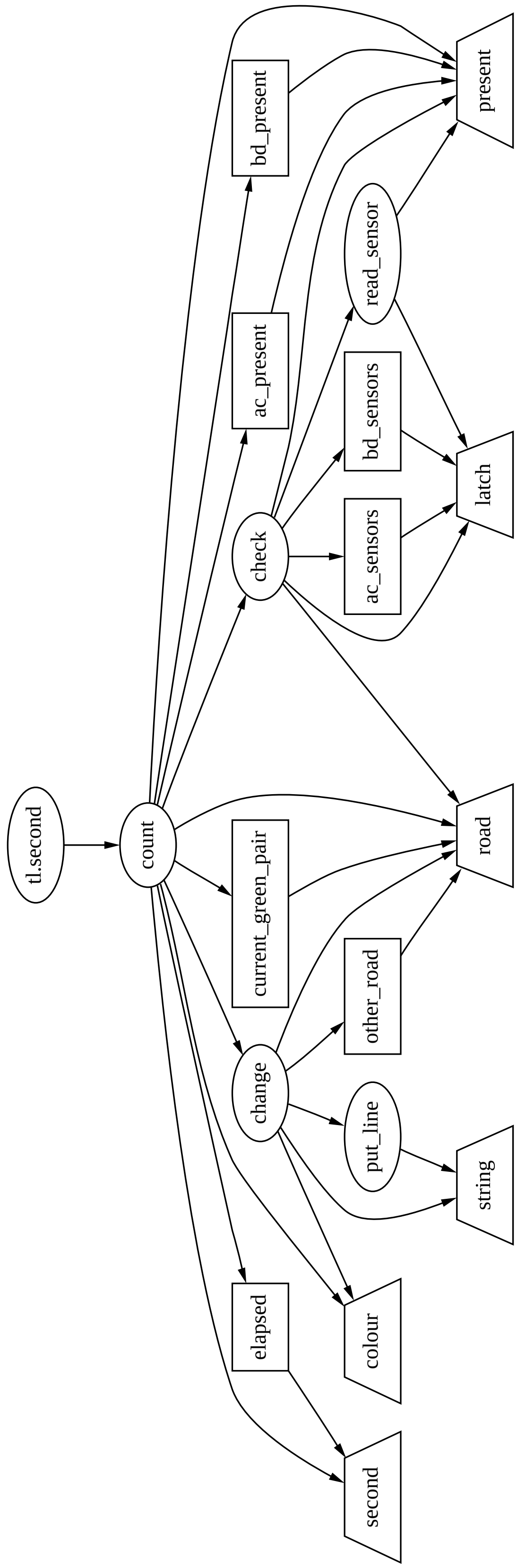
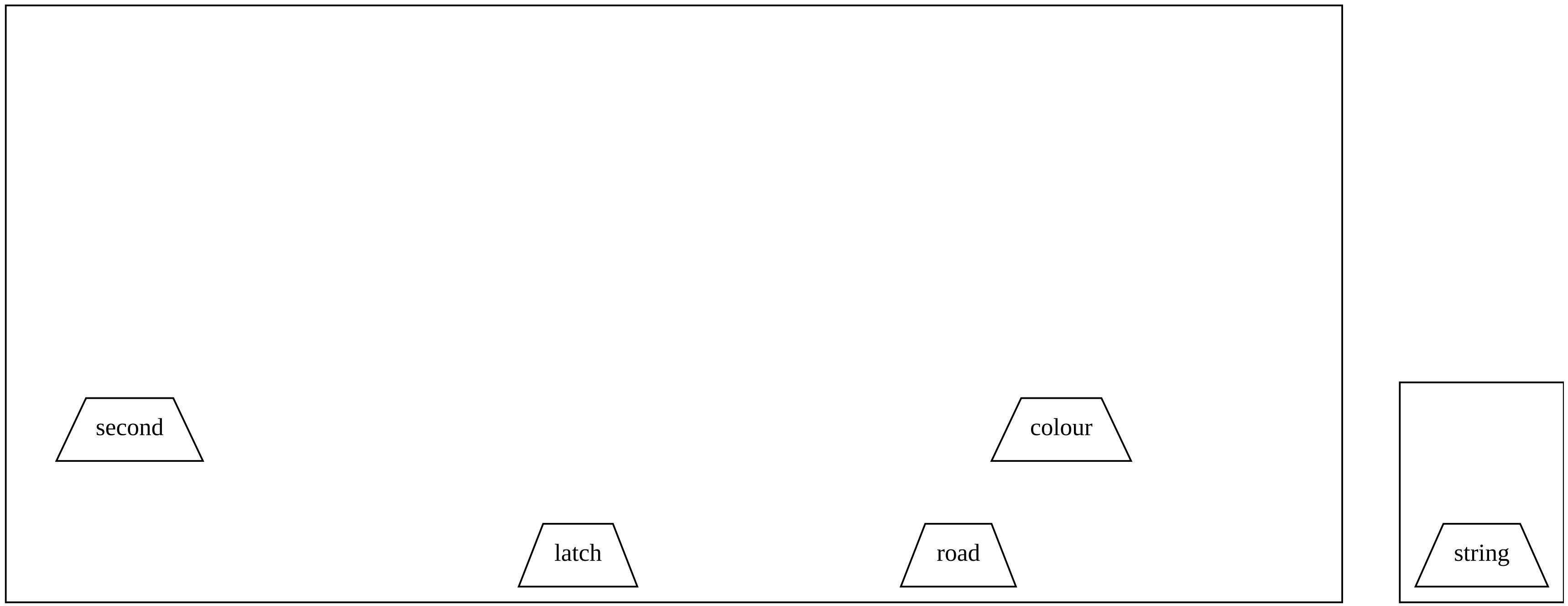


Figure 8.28: Graph of flat Traffic Light design without environment


second
colour
latch
road
string

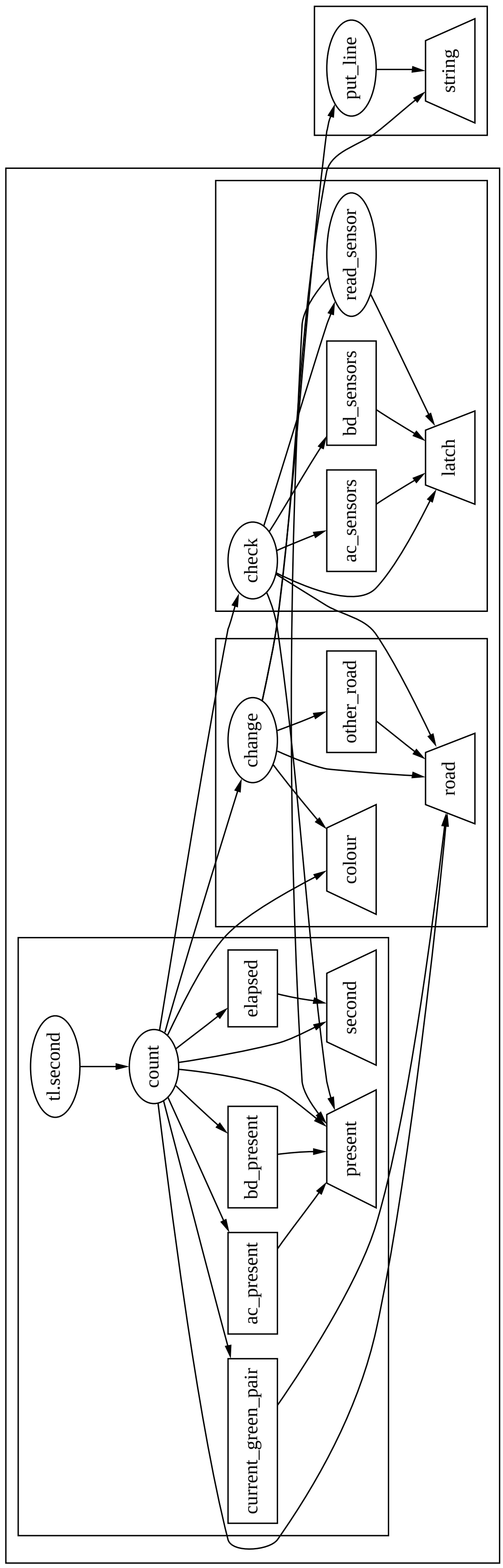


Figure 8.31: Graph of *orp us*'s Traffic Light design with environment

- The other interesting change at the module level was the splitting of the spreadsheet calculation pad into two separate but equal objects, one responsible for validating potential commands, the other for actually performing the required changes.

  Much of the internal complexity of the calculation pad comes from the need to propagate changes to dependent slots and to save the spreadsheet in a suitable order for later restoration. By splitting the validation from the implementation, the mechanics for propagating changes can be further encapsulated.

8.3.1 Improvement?

Full details of both the initial and post-processing modular structures are shown in Section B.3. The original design had a complexity of 12038.9 bits, whilst *orp us*'s suggestion had a complexity of 9618.8 bits, a saving of about 20 percent.

Redistributing the datatypes is clearly a good thing, as the majority of types are only used locally inside a module. We choose to separate them partly fo required. The editor would not require any changes and basic validation would remain unchanged since it can already check for valid slot identities and slot containing strings rather than arithmetical expressions.

The calculation pad would need a look-up table for validation and despatch to the appropriate operation. However, the basic entry point for updating the calculation pad would remain unchanged.

Chapter 9

Summary and Conclusion

Synopsis

In this chapter, we present a critical evaluation of this thesis and its contribution to knowledge. Section 9.1 provides a brief summary of this thesis. Section 9.2 discusses what this thesis has achieved, and how far its objectives have been met. Section 9.3 looks forward to future work, as a result of this research. Finally, Section 9.4 provides a brief overall conclusion.

9.1 Summary of this Thesis

This thesis has studied the problem of providing an intelligent system to aid software designers improve the quality of their designs. We have limited ourselves to the objective evaluation of a design's modular complexity. Little previous work has been done in producing systems for even this limited objective.

Although there are clearly many other factors which influenc

9.2 Evaluation

The previous section provided a brief summary of this thesis. In this section we shall look at how well this work meets our original objective.

9.2.1 Achievements

There can be no doubt that we have created a prototype tool which takes in an architectural design expressed in HOOD, and finds alternative designs with less complex structure. Complexity has been defined in terms of the length of a message describing the structure of the design. The use of message length as a measure of complexity is founded on Kolmogorov complexity, which gives us an objective basis for comparing the structural complexity of designs.

We have shown that our complexity metric satisfies criteria that other researchers have suggested are good properties for complexity measures. This has been rigorously proved for the

- A better understanding of the relationship between coupling and cohesion. Particularly in situations were the worst forms of coupling are not permitted.

As part of the development of *orp us*, we have identified a number of deficiencies in HOOD as currently defined. Some of these deficiencies we have addressed in our extensions to HOOD. The others (e.g., nested operations) would be quite simple to add, but require further consideration because they alter what may be regarded as the philosophy of HOOD.

Although *orp us* is based on HOOD, in principle there is no reason why (with suitable

- Alternative theories for modelling a hierarchical graph need to be investigated, and their impact on message lengths determined. This thesis assumes a single class of theories for describing a hierarchical graph. There are undoubtedly others, some of which may yield smaller message lengths and thus more closely approximate the true[3] Kolmogorov Complexity of the underlying design.

- Providing a clearer method for reporting *orp us*'s results, in a manner readily understandable to the end-user. *orp us*'s output is currently rather cryptic, and not obviously related to the initial design; particularly as module names are not preserved. To make *orp us* acceptable in an industrial setting, a simple to understand output is required. Even better would be to reverse engineer *orp us*'s output into a design notation; ideally using the same notation as the original design.

- Integrating *orp us* into other CASE tools. *orp us* is really intended as the back-end of a CASE tool, and not for direct use by a designer. We need to merge *orp us* into a CASE tool so that it has access to other facilities (in particular a database for storing large designs) and supports industrial use.

9.4 Contribution of This Thesis

The research reported in this thesis has developed a metric for measuring the absolute complexity of a software design's architecture. Complexity is measured in an objective manner and does not

Halstead, M. H. (1977). ***Elements of Software Science***. Elsevier North-Holland, New York, NY.

Hardy, G. H. (1947). ***A Mathematican's Apology***. Cambridge University Press.

Harrison, W. (1992). An entropy-based measure of software complexity. ***IEEE Transactions on Software Engineering***, 18(11):1025–1029.

Harrison, W. and Ossher, H. (1993). Subject-oriented programming (a critique of pure objects). In ***OOPSLA '93: Eighth annual conference on Object-Oriented Programming Systems, Languages and Applications***, pages 411–428, Washington, DC. ACM Press. published in ACM SIGPLAN Notices 28(10).

Henry, S. and Kafura, D. (1993). The evaluation of software systems' structure using quantitative software metrics. In Shepperd, M. J., editor, ***Software Engineering Metrics, Volume 1: Measures and Validations***, McGraw-Hill international series in software engineering, chapter 6, pages 99–111. McGraw-Hill Book Company, Maidenhead, England. Reprinted from Software Practice and Experience, 14(6); 561–573, 1984.

Hoare, C. A. R. (1985). ***Communicating Sequential Processes***. Prentice-Hall.

HOOD HRM (1995). ***HOOD Reference Manual, Release 4***. HOOD Technical Group. HRM4–6/292.M) . .H44021())4.23177(.)-2(,)-330.83(W)89.1905(.)21.8485(n)6wHo-, itiie

Khoshgoftaar, T. M. and Allen, E. B. (1994). Applications of information theory to software engineering measurement. ***Software Quality Journal***, 3(2):79–103.

Koutsofios, E. and North, S. C. (1994). ***Editing graphs with*** dotty. AT&T Bell Laboratories, Murray Hill, NJ.

Laventhol, J. (1987). ***Programming in POP-11***. Blackwell Scientific Publications Ltd.

Lew, K. S., Dillon, T. S., and Forward, K. E. (1988). Software complexity and its impact on software reliability. ***IEEE Transactions on Software Engineering***, 14(11):1645–1655.

Li, M. and Vitányi, P. (1993). ***An Introduction to Kolmogorov Complexity and its Applications***. Texts and Monographs in Computer Science. Springer-Verlag, New York, NY.

Li, M. and Vitányi, P. (1997). ***An Introduction to Kolmogorov Complexity and its Applications***. Graduate texts in Computer Science. Springer-Verlag, New York, NY, 2nd edition.

Listov, B. and Guttag, J. (1986). ***Abstraction and Specification in Program Development***, pages 433–444. The MIT Press, Cambridge, MA.

Lor, K. W. E. and Berry, D. M. (1991). Automatic synthesis of SARA design models from system requirements. ***IEEE Transactions on Software Engineering***, 17(12):1229–1240.

MacKay, D. J. C. (1997). Information theory, pattern recognition and neural networks. forthcoming, available in http://wol.ra.phy.cam.ac.uk/mackay/itprnn.

Marca, D. A. and McGowan, C. L. (1988). ***SADT—Structured Analysis and Design Technique***. McGraw-Hill.

MASCOT (1987). ***The Official Handbook of MASCOT: Version 3.1***. Her Majesty's Stationery Office, London, England. Joint IECCA and MUF Committee on MASCOT.

McCabe, T. J. (1976). A software complexity measure. ***IEEE Transactions on Software Engineering***, 2(6):308–320.

McCabe, T. J. and Butler, C. W. (1989). Design complexity measurements and testing. ***Communications of the ACM***, 32(12):1415–1425.

Oliver, J. J. and Hand, D. J. (1994). Introduction to minimum encoding inference. Technical Report 4–94, Department of Statistics, Open University, England. revised Dec. 1996.

Orr, K. T. (1971). ***Structured System Development***. Yourdon Press, New York, NY.

Page-Jones, M. (1988). ***The Practical Guide to Structured Systems Design***. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition.

Page-Jones, M. (1992). Comparing techniques by means of encapsulation and connascence. ***Communications of the ACM***, 35(9):147–151.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. ***Communications of the ACM***, 15(12):1053–1058.

Parnas, D. L. (1997). Software engineering: An unconsummated marriage. ***Communications of the ACM***, 40(9):128.

Popper, K. R. (1968). ***The Logic of Scientific Discovery***. Harper and Row, New York, NY.

Pressman, R. S. (1992). ***Software Engineering: A Practitioner's Approach***. McGraw-Hill Inc., New York, NY, 3rd edition.

Rayward-Smith, V. J. (1983).

Shepperd, M. J. and Ince, D. C. (1993). ***Derivation and Validation of Software Metrics***. The International Series of Monographs on Computer Science. Clarendon Press, Oxford, England.

Simon, H. A. (1973). The structure of ill structured problems. ***Artifical Intelligence***, 4:181–200.

Simon, H. A. (1976). ***Administrative Behaviour***. The Free Press, New York, NY, 3rd edition.

Simon, H. A. (1981). ***The Sciences of the Artifical***. The MIT Press, Cambridge, MA, 2nd edition.

Spivey, J. M. (1989). ***The Z Notation: A Reference Manual***. Prentice-Hall, Hemel Hempstead, England.

Stroustrup, B. (1994). ***The C++ Programming Language***. Addison-Wesley Publishing Company, Reading, MA, 2nd edition.

Thornton, C. J. and du Boulay, B. (1992).

Appendix A

Augmented HOOD

This appendix presents the changes to HOOD's Standard Interchange Format, as documented in Delatte et al. (1993, appendix D). The syntax is presented using BNF notation as described in Delatte et al. (1993). Meta-comments are delimited by '/*' and '*/'. Numbers in round brackets refer to the syntax phrases defined in Delatte et al..

A.1 Changes to Existing Syntax

A.1.1 Pseudo Code

pseudo_code_section ::= /* (29) */
    PSEUDO_CODE
        [code_linkage_section]
        [free_text]
| **PSEUDO_CODE**
        NONE

A.2 Pseudo Code Enhancements

code_linkage_section ::=
    OPERATION_REQUIREMENTS
        [requires_type_section]
        [reads_from_section]
        [writes_to_section]
    END_OPERATION_REQUIREMENTS
| **OPERATION_REQUIREMENTS**
        NONE

requires_type_section ::=
    REQUIRES_TYPE
        type_reference semi_colon
        {***type***_reference semi_colon}
| **REQUIRES_TYPE**
        NONE

reads_from_

Elements of the storage area may be of three kinds: ***blank***, ***value*** or ***comment***.

- A blank line has no value.
- A comment is a string of up to eight characters.[9]
- When a comment occupies an element of storage, it has no effect on any other element of storage.

A value is a floating point number.[10] Values in storage may be related to each other by multiplication, division, addition and subtraction. These relations are set by the user and may be changed at any time. Some values in storage will be entered by the user as constants or parameters; others will be derived as a result of one of the relations mentioned a property that if they were executed in sequence, beginning with a blank storage area, they would generate the storage area in effect at the time the save was done.[15]

`LOAD;FILE:`***na e*****;** The current state of the storage area is discarded and then reloaded based on the contents of the file named ***name.tc***. The file is assumed to be in the format produced by the `AVE` command.

`TORE-COMMENT;WITH:`***string*****;**`AT:`***slot-address*****;** The comment ***string*** is stored in the element of storage labelled by ***slot-address***.

`TORE-VALUE;WITH:`***nu ber*****;**`AT:`***slot-address*****;** The value ***number***[16] is stored in the element labelled by ***slot-address***. After this command has been executed, this element will not depend on any other elements.

`BLANK; LOT:`***slot-address*****;** The element labelled by ***slot-address*** becomes blank.

`QUIT;` The execution of *TriviCalc* is terminated and control is returned to the executive.

Movement Commands

UBTRACT **or** $-$ The element of storage labelled ***slot3*** is related to the other two elements as (***slot1***$-$***slot2***).

MULTIPLY **or** $\star$ The element of storage labelled ***slot3*** is related to the other two elements as (***slot1***$\star$***slot2***).

DIVIDE **or** / The element of storage labelled ***slot3*** is related to the other two elements as (***slot1***/***slot2***).[20]

The Working-Area Editor

The working-area editor is a simple modeless editor with special functions to simplify the input of commands to the *TriviCalc* command processor. The editor maintains a cursor in the working area. Every keystroke is considered to be a command to the editor. All commands are atomic; they are either processed to completion immediately or halt in error, after doing nothing except possibly displaying an error message. Some keystrokes denote textual values (the characters, numerals and punctuation keys). The command that is run by typing any of these keystrokes merely inserts the key's textual value at the cursor.[21] These are known as ***textual input commands***.

Other keystrokes do not denote textual values. These are special keys (such the carriage-return or delete), or are typed by holding down the CONTROL key and pressing some other key. These non textual keystrokes are interpreted by the editor as commands that affect the text in the working area. A brief description of the nontextual commands follows.

CONTROL L Move the cursor to the left one position.

CONTROL R Move the cursor to the right one position.

CONTROL D Delete the character at the cursor, if there is one.

DELETE Delete the character to the left of the cursor, if there is one.

CONTROL A Operator Adjust. If the working area is of the form

slot1 op slot2

where ***op*** is one of the characters $\star$, /, $-$ or $+$ and ***slot1***, ***slot2*** are strings, the contents of the working area are replaced with

op;VALUE1:***slot1***;VALUE2:***slot2***;GIVING:%;

Otherwise, if the contents of the working area represent a valid numerical value, the working area is interpreted as ***number*** and its contents replaced by

TORE-VALUE;WITH:***number***;AT:%;

Otherwise, the working area is interpreted as ***string*** and its contents are replaced by

TORE-COMMENT;WITH:***string***;AT:%;

Finally, the effect of a CONTROL K command with the cursor at the beginning of the working area, followed by a CONTROL E command is simulated. The effect of this is to replace the % character with the address of the current slot.

[20] What happens if the value of ***slot2***

CONTROL K Search from the position to the right for a %, wrapping around to the beginning of the working area if the end of the working area is reached. If a % is found, delete it and leave the cursor at its position. If none is found, do nothing.[22]

CONTROL E The address of the current slot in the storage area is inserte

```
      close_file ( channel : IN channel ) ;
      matches ( pattern : IN list ; datum : IN list ) RETURN boolean ;
      parse_string ( text : IN string ) RETURN list ;
      isstring ( text : IN string ) RETURN boolean ;
      read_line ( channel : IN channel ) RETURN string ;
      get_input_char ( channel : IN channel ) RETURN character ;
      write_line ( channel : IN channel ; text : IN string ) ;
      sysexit ;
END_OBJECT pop_11



OBJECT trivicalc IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      main_program ;
  INTERNALS
    OBJECTS
      cli ;
      data_types ;
      dm ;
      em ;
      sa ;
      wae ;

    OPERATIONS
      main_program
        IMPLEMENTED_BY cli.main_program ;
END_OBJECT trivicalc



OBJECT wae IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      editor ;
      get_cs RETURN slot_id ;
      init_cl ;
      init_cs ;
      is_macro_name ( id : IN string ) RETURN boolean ;
      recall_all_macros RETURN list_strings ;
      set_cs ( slot : IN slot_id ) ;
      store_macro ( id : IN integer ; text : IN string ) ;

  REQUIRED_INTERFACE
    OBJECT cli
      OPERATIONS
        command_despatcher ( command : IN string ) RETURN validity
    OBJECT dm
      OPERATIONS
        delete_char ;
        delete_char_at_left ;
        delete_line ;
        display_cl_line ( text : IN string ) ;
        insert_char ( char : IN character ) ;
        insert_string ( text : IN string ) ;
        move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
        position_cl_cursor ( cursor : IN cursor_position ) ;
        ring_bell ;
```

```
        set_cs ( slot : IN slot_id ) ;
    OBJECT em
      OPERATIONS
        escape_seen ;
    OBJECT sa
      OPERATIONS
        get_contents ( slot : IN slot_id ) RETURN content ;

  INTERNALS
    OBJECTS
      cl ;  ;;; command line
      cp ;  ;;; command processor
      cs ;  ;;; current slot
      il ;  ;;; internal locations
    OPERATIONS
      editor
        IMPLEMENTED_BY cp.editor ;
      init_cl
        IMPLEMENTED_BY cl.init_cl ;
      init_cs
        IMPLEMENTED_BY cs.init_cs ;
      recall_all_macros RETURN list_strings
        IMPLEMENTED_BY il.recall_all_macros RETURN list_strings ;
      set_cs ( slot : IN slot_id )
        IMPLEMENTED_BY cs.set_cs ( slot : IN slot_id ) ;
      get_cs RETURN slot_id
        IMPLEMENTED_BY cs.get_cs RETURN slot_id ;
      is_macro_name ( id : IN string ) RETURN boolean
        IMPLEMENTED_BY il.is_macro_name
                    ( id : IN string ) RETURN boolean ;
      store_macro ( id : IN integer ; text : IN string )
        IMPLEMENTED_BY il.store_macro ( id : IN integer ;
                                        text : IN string ) ;
END_OBJECT wae


OBJECT cs IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      get_cs RETURN slot_id ;
      get_cs_content RETURN content ;
      init_cs ;
      move_cs_down ;
      move_cs_left ;
      move_cs_right ;
      move_cs_up ;
      set_cs ( slot : IN slot_id ) ;

  REQUIRED_INTERFACE
    OBJECT dm
      OPERATIONS
        move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
        set_cs ( slot : IN slot_id ) ;
    OBJECT sa
      OPERATIONS
        get_contents ( slot : IN slot_id ) RETURN content ;

  INTERNALS
```

```
OPERATIONS
  get_cs RETURN slot_id ;
  get_cs_content RETURN content ;
  init_cs ;
  move_cs_down ;
  move_cs_left ;
  move_cs_right ;
  move_cs_up ;
  set_cs ( slot : IN slot_id ) ;
DATA
  current_slot : slot_id ;


OPERATION_CONTROL_STRUCTURES

  OPERATION move_cs_up
    USED_OPERATIONS
      dm.move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
      min_row ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          current_slot ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION move_cs_up

  OPERATION move_cs_down
    USED_OPERATIONS
      dm.move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
      max_row ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          current_slot ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION move_cs_down

  OPERATION move_cs_left
    USED_OPERATIONS
      dm.move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
      min_column ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          current_slot ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION move_cs_left

  OPERATION move_cs_right
    USED_OPERATIONS
      dm.move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
      max_column ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          current_slot ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION move_cs_right
```

```
OPERATION get_cs RETURN slot_id
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      READS_FROM
        current_slot ;
    END_OPERATION_REQUIREMENTS
END_OPERATION
```

```
REQUIRED_INTERFACE
  OBJECT dm
    OPERATIONS
      delete_char ;
      delete_char_at_left ;
      delete_line ;
      display_cl_line ( text : IN string ) ;
      insert_char ( char : IN character ) ;
      insert_string ( text : IN string ) ;
      position_cl_cursor ( cursor : IN cursor_position ) ;
      ring_bell ;


INTERNALS
  CONSTANTS
    max_cursor : integer ;
    max_length : integer ;
    min_cursor : integer ;
    min_length : integer ;
  OPERATIONS
    cursor_left ;
    cursor_right ;
    delete_char ;
    delete_char_left ;
    delete_line ;
    get_cl RETURN string ;
    init_cl ;
    insert_char ( char : IN character ) ;
    insert_string ( text : IN string ) ;
    locate_sol ;
    replace_percent RETURN validity ;
  DATA
    cursor : integer ;
    eos  : integer ;
    line   : string ;

  OPERATION_CONTROL_STRUCTURES

    OPERATION cursor_left
      USED_OPERATIONS
        dm.position_cl_cursor ( cursor : IN cursor_position ) ;
        min_cursor ;
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          WRITES_TO
            cursor ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION cursor_left

    OPERATION cursor_right
      USED_OPERATIONS
        dm.position_cl_cursor ( cursor : IN cursor_position ) ;
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          READS_FROM
            eos ;
          WRITES_TO
            cursor ;
```

```
    END_OPERATION_REQUIREMENTS
END_OPERATION cursor_right

OPERATION insert_char ( char : IN character )
  USED_OPERATIONS
    dm.insert_char ( char : IN character ) ;
    dm.ring_bell ;
    max_length ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      WRITES_TO
        cursor ;
        eos ;
        line ;
    END_OPERATION_REQUIREMENTS
END_OPERATION insert_char

OPERATION insert_string ( text : IN string )
  USED_OPERATIONS
    dm.position_cl_cursor ( cursor : IN cursor_position ) ;
    dm.display_cl_line ( text : IN string ) ;
    dm.ring_bell ;
    max_length ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      WRITES_TO
        cursor ;
        eos ;
        line ;
    END_OPERATION_REQUIREMENTS
END_OPERATION insert_string

OPERATION delete_char
  USED_OPERATIONS
    dm.delete_char ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      READS_FROM
        cursor ;
      WRITES_TO
        eos ;
        line ;
    END_OPERATION_REQUIREMENTS
END_OPERATION delete_char

OPERATION delete_char_left
  USED_OPERATIONS
    dm.delete_char_at_left ;
    min_cursor ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      WRITES_TO
        cursor ;
        eos ;
        line ;
    END_OPERATION_REQUIREMENTS
END_OPERATION delete_char_left
```

```
      OPERATION delete_line
        USED_OPERATIONS
          dm.delete_line ;
          min_cursor ;
          min_length ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              cursor ;
              eos ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION delete_line

      OPERATION replace_percent RETURN validity
        USED_OPERATIONS
          dm.position_cl_cursor ( cursor : IN cursor_position ) ;
          delete_char ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            REQUIRES_TYPE
              string ;
              boolean ;
            READS_FROM
              cursor ;
              eos ;
              line ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION replace_percent

      OPERATION locate_sol
        USED_OPERATIONS
          dm.position_cl_cursor ( cursor : IN cursor_position ) ;
          min_cursor ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              cursor ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION locate_sol

      OPERATION get_cl RETURN string
        USED_OPERATIONS
          min_length ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            READS_FROM
              cursor ;
              eos ;
              line ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION get_cl

      OPERATION init_cl
        USED_OPERATIONS
          delete_line ;
      END_OPERATION init_cl

END_OBJECT cl
```

```
OBJECT il IS PASSIVE
  PROVIDED_INTERFACE
    CONSTANTS
      old_cl : integer ;
    OPERATIONS
      is_macro_name ( id : IN string ) RETURN boolean ;
      store_macro ( id : IN integer ; text : IN string ) ;
      recall_macro ( id : IN integer ) RETURN string ;

  INTERNALS
    CONSTANTS
      max_macro : integer ;
      min_macro : integer ;
      old_cl    : integer ;
    OPERATIONS
      store_macro ( id : IN integer ; text : IN string ) ;
      recall_macro ( id : IN integer ) RETURN string ;
      recall_all_macros RETURN list_strings ;
      is_macro_name ( id : IN string ) RETURN boolean ;
      init_il ;
    DATA
      macros : string ;

    OPERATION_CONTROL_STRUCTURES

      OPERATION store_macro ( id : IN integer ; text : IN string )
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              macros ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION store_macro

      OPERATION recall_macro ( id : IN integer ) RETURN string
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            READS_FROM
              macros ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION recall_macro

      OPERATION recall_all_macros RETURN list_strings
        USED_OPERATIONS
          max_macro ;
          min_macro ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            READS_FROM
              macros ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION recall_all_macros

      OPERATION is_macro_name ( id : IN string ) RETURN boolean
        USED_OPERATIONS
          max_macro ;
          min_macro ;
      END_OPERATION is_macro_name
```

```
      OPERATION init_il
        USED_OPERATIONS
          max_macro ;
          min_macro ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              macros ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION init_il

END_OBJECT il



OBJECT cp IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      editor ;

  REQUIRED_INTERFACE
    OBJECT cli
      OPERATIONS
        command_despatcher ( command : IN string ) RETURN validity
  INTERNALS
    OBJECTS
      editor ;  ;;; main editor
      ccp ;     ;;; Control Character Processing
    OPERATIONS
      editor
        IMPLEMENTED_BY editor.editor ;
END_OBJECT cp



OBJECT editor IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      editor ;

  REQUIRED_INTERFACE
    OBJECT ccp
      OPERATIONS
        process_control_char ( char : IN character ) ;
        is_control_char ( char : IN character ) RETURN boolean ;
    OBJECT dm
      OPERATIONS
        ring_bell ;
    OBJECT em
      OPERATIONS
        escape_seen ;
    OBJECT pop_11
      OPERATIONS
        get_input_char ( channel : IN channel ) RETURN character ;
        open ( file_name : IN string ; mode : IN string ) RETURN channel ;
        close_file ( channel : IN channel ) ;

  INTERNALS
```

```
OPERATIONS
  editor ;
  get_char RETURN character
```

```
                channel ;
            END_OPERATION_REQUIREMENTS
        END_OPERATION term_cp

END_OBJECT editor


OBJECT ccp IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      process_control_char ( char : IN character ) ;
      is_control_char ( char : IN character ) RETURN boolean ;
  REQUIRED_INTERFACE
    OBJECT cl
      OPERATIONS
        cursor_left ;
        cursor_right ;
        get_cl RETURN string ;
        delete_line ;
        delete_char ;
        delete_char_left ;
        insert_string ( text : IN string ) ;
        locate_sol ;
        replace_percent RETURN validity ;
    OBJECT cs
      OPERATIONS
        move_cs_down ;
        move_cs_left ;
        move_cs_edsa_cs_left


OBJErEe_cs_edsa_cs_left:
```

```
CONSTANTS
  despatch_table : pop_11.property_table ;
OPERATIONS
  ccp_adjust ;
  ccp_cli ;
  ccp_cli_keep ;
  ccp_delete_char ;
  ccp_delete_char_left ;
  ccp_delete_line ;
  ccp_get_cs ;
  ccp_get_cs_content ;
  ccp_recall_macro ;
  ccp_replace_percent ;
  ccp_store_macro ;
  is_control_char ( char : IN character ) RETURN boolean ;
  obey_cl RETURN validity ;
  process_control_char ( char : IN character ) ;
  replace_cl ( text : IN string ) ;
  save_cl ;

OPERATION_CONTROL_STRUCTURES

  OPERATION is_control_char ( char : IN character ) RETURN boolean
    USED_OPERATIONS
      despatch_table ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        REQUIRES_TYPE
          pop_11.property_table ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION is_control_char

  OPERATION process_control_char ( char : IN character )
    USED_OPERATIONS
      despatch_table ;
      ccp_adjust ;
      ccp_cli ;
      ccp_cli_keep ;
      ccp_delete_char ;
      ccp_delete_char_left ;
      ccp_delete_line ;
      ccp_get_cs ;
      ccp_get_cs_content ;
      ccp_recall_macro ;
      ccp_replace_percent ;
      ccp_store_macro ;
      cl.cursor_left ;
      cl.cursor_right ;
      cs.move_cs_down ;
      cs.move_cs_left ;
      cs.move_cs_right ;
      cs.move_cs_up ;
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        REQUIRES_TYPE
          pop_11.property_table ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION process_control_char
```

```
OPERATION save_cl
  USED_OPERATIONS
    il.store_macro ( id : IN integer ; text : IN string ) ;
    il.old_cl ;
    cl.get_cl RETURN string ;
END_OPERATION save_cl

OPERATION obey_cl RETURN validity
  USED_OPERATIONS
    cl.get_cl RETURN string ;
```

```
    cs.get_cs RETURN slot_id ;
    em.report_error ( text : IN string ) ;
    sa.is_comment ( text : IN string ) RETURN boolean ;
    sa.is_float ( text : IN string ) RETURN boolean ;
    sa.is_operation ( text : IN string ) RETURN boolean ;
    sa.is_slot ( text : IN string ) RETURN boolean ;
    pop_11.matches ( pattern : IN list ;
                     datum : IN list ) RETURN boolean ;
    replace_cl ( text : IN string ) ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        float ;
        operation_math ;
        s_string ;
    END_OPERATION_REQUIREMENTS
END_OPERATION ccp_adjust

OPERATION ccp_replace_percent
  USED_OPERATIONS
    cl.replace_percent
```

```
END_OBJECT ccp


OBJECT cli IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      command_despatcher ( command : IN string ) RETURN validity
      main_program ;

  REQUIRED_INTERFACE
    OBJECT dm
      OPERATIONS
        init_dm ;
    OBJECT em
      OPERATIONS
        init_em ;
        report_error ( text : IN string ) ;
    OBJECT sa
      OPERATIONS
        init_sa ;
        is_comment ( text : IN string ) RETURN boolean ;
        is_float ( text : IN string ) RETURN boolean ;
        is_operation ( text : IN string ) RETURN boolean ;
        is_slot ( text : IN string ) RETURN boolean ;
        save_sa RETURN list_strings ;
        set_slot ( slot : IN slot_id ;
                   value : IN content ) RETURN validity ;
    OBJECT wae
      OPERATIONS
        editor ;
        get_cs RETURN slot_id ;
        init_cl ;
```

```
set_current_slot ( command : IN
```

```
      END_OPERATION reinitialise_system

      OPERATION main_program
        USED_OPERATIONS
          em.init_em ;
          reinitialise_system ;
          wae.editor ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              load_in_progress ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION main_program

END_OBJECT cli



OBJECT sa IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      get_contents ( slot : IN slot_id ) RETURN content ;
      init_sa ;
      is_comment ( text : IN string ) RETURN boolean ;
      is_float ( text : IN string ) RETURN boolean ;
      is_operation ( text : IN string ) RETURN boolean ;
      is_slot ( text : IN string ) RETURN boolean ;
      save_sa RETURN list_strings ;
      set_slot ( slot : IN slot_id ; value : IN content )
                RETURN validity ;

  REQUIRED_INTERFACE
    OBJECT dm
      OPERATIONS
        display_value ( slot : IN slot_id ) ;
    OBJECT em
      OPERATIONS
        report_error ( text : IN string ) ;

  INTERNALS
    CONSTANTS
      blank      : slot_type ;
      comment    : slot_type ;
      expression : slot_type ;
      float      : slot_type ;
    OPERATIONS
      get_contents ( slot : IN slot_id ) RETURN content ;
      init_sa ;
      is_comment ( text : IN string ) RETURN boolean ;
      is_float ( text : IN string ) RETURN boolean ;
      is_operation ( text : IN string ) RETURN boolean ;
      is_slot ( text : IN string ) RETURN boolean ;
      save_sa RETURN list_strings ;
      set_slot ( slot : IN slot_id ;
                 value : IN content ) RETURN validity ;
      blank ( slot : IN slot_id ) ;
      address ( slot : IN slot_id ) RETURN slots_index ;
      create_sa ;
      create_slot ( column : IN column_position ;
```

```
                row : IN row_position ) RETURN a_slot ;
  add_successor ( slot : IN slot_id ; to_slot : IN slot_id ) ;
  remove_successor ( slot : IN slot_id ; from_slot : IN slot_id ) ;
  list_successors ( slot : IN slot_id ) RETURN list_slot_ids ;
  is_successor ( slot : IN slot_id ;
                 of_slot : IN slot_id ) RETURN boolean ;
  complete_update ( slot : IN slot_id ; success : IN boolean ) ;
  display_value ( slot : IN slot_id ) ;
  depth_first_search ( slot : IN slot_id ) RETURN slot_id ;
  update_order ( slot : IN slot_id ) RETURN list_slot_ids ;
  update_slots ( slots : IN list_slot_ids ) RETURN validity ;
  evaluate ( slot : IN slot_id ) RETURN full_value ;
  is_slot_arithmetic ( slot : IN slot_id ) RETURN boolean ;
  is_slot_float ( slot : IN slot_id ) RETURN boolean ;
  is_slot_blank ( slot : IN slot_id ) RETURN boolean ;
  is_slot_comment ( slot : IN slot_id ) RETURN boolean ;
  is_slot_expression ( slot : IN slot_id ) RETURN boolean ;
  get_value ( slot : IN slot_id ;
              new_value : IN boolean ) RETURN value ;
DATA
  slots : slot_array ;
  stack : list_slot_ids ;
```

```
    blank ( slot : IN slot_id ) ;
    create_sa ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        row_position ;
        column_position ;
    END_OPERATION_REQUIREMENTS
END_OPERATION init_sa

OPERATION create_sa
  USED_OPERATIONS
    create_slot ( column : IN column_position ;
                  row : IN row_position ) RETURN a_slot ;
    min_letter ;
    max_letter ;
    min_row ;
    max_row ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        slot_id ;
      WRITES_TO
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION create_sa

OPERATION create_slot ( column : IN column_position ;
                        row : IN row_position ) RETURN a_slot
  USED_OPERATIONS
    blank ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        boolean ;
        list ;
        slot_type ;
    END_OPERATION_REQUIREMENTS
END_OPERATION create_slot

OPERATION add_successor ( slot : IN slot_id ; to_slot : IN slot_id )
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        list ;
        list_slot_ids ;
      WRITES_TO
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION add_successor

OPERATION remove_successor ( slot : IN slot_id ;
                             from_slot : IN slot_id )
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        list ;
        list_slot_ids ;
```

```
      WRITES_TO
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION remove_successor

OPERATION list_successors ( slot : IN slot_id ) RETURN list_slot_ids
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        list ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION list_successors

OPERATION is_successor ( slot : IN slot_id ;
                         of_slot : IN slot_id ) RETURN boolean
  USED_OPERATIONS
    list_successors ( slot : IN slot_id
```

```
      WRITES_TO
        stack ;
    END_OPERATION_REQUIREMENTS
END_OPERATION depth_first_search

OPERATION update_order ( slot : IN slot_id ) RETURN list_slot_ids
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
    depth_first_search ( slot : IN slot_id ) RETURN slot_id ;
    list_successors ( slot : IN slot_id ) RETURN list_slot_ids ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
```

```
    address ( slot : IN slot_id ) RETURN slots_index ;
    blank ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        a_slot ;
        slot_type ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION is_slot_blank

OPERATION is_slot_comment ( slot : IN slot_id ) RETURN boolean
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
    comment ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        a_slot ;
        slot_type ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION is_slot_comment

OPERATION is_slot_float ( slot : IN slot_id ) RETURN boolean
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
    float ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        a_slot ;
        slot_type ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION is_slot_float

OPERATION is_slot_expression ( slot : IN slot_id ) RETURN boolean
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
    expression ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        a_slot ;
        slot_type ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION is_slot_expression

OPERATION is_slot_arithmetic ( slot : IN slot_id ) RETURN boolean
  USED_OPERATIONS
    is_slot_expression ( slot : IN slot_id ) RETURN boolean
    is_slot_float ( slot : IN slot_id ) RETURN boolean
```

```
    END_OPERATION_REQUIREMENTS
END_OPERATION is_operation

OPERATION is_float ( text : IN string ) RETURN boolean
  USED_OPERATIONS
    isstring ( text : IN string ) RETURN boolean ;
    max_float ;
    min_float ;
END_OPERATION is_float

OPERATION is_comment ( text : IN string ) RETURN boolean
  USED_OPERATIONS
    isstring ( text : IN string ) RETURN boolean ;
    max_s_string ;
END_OPERATION is_comment

OPERATION get_value ( slot : IN slot_id ;
                      new_value : IN boolean ) RETURN value
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        slots_index ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION get_value

OPERATION get_contents ( slot : IN slot_id ) RETURN content
  USED_OPERATIONS
    address ( slot : IN slot_id ) RETURN slots_index ;
  PSEUDO_CODE
    OPERATION_REQUIREMENTS
      REQUIRES_TYPE
        slots_index ;
      READS_FROM
        slots ;
    END_OPERATION_REQUIREMENTS
END_OPERATION get_contents

OPERATION save_sa RETURN
```

```
EE01.0s0a1000T(sla.o.et_value)Tj2.310Td(()Tjrtsla.o.es0a100212Tde.tevcwea
END_OPERATION_REQUIREMENTS
li_0g1000w2.020TdTd(ENDd(0021DIONSDNS)Tj4y0400ION_REQUIRED)UTj84DDaNS)D2oTd
ss ( slot
EE01.0s0a1000T
```

```
            column_position ;
            list ;
            list_slot_ids ;
            list_strings ;
            slots_index ;
          READS_FROM
            slots ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION save_sa

END_OBJECT sa


OBJECT dm IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      clear_error_display ;
      display_error_message ( text : IN string ) ;
      display_value ( slot : IN slot_id ) ;
      move_cs ( old_cs : IN slot_id ; cs : IN slot_id ) ;
      set_cs ( cs : IN slot_id ) ;
      delete_char ;
      delete_char_at_left ;
      delete_line ;;I_line        ( text : IN string ) ;
                  insert_char ( char : IN character ) ;
                  insert_string ( text : IN string ) ;;position(I_curso)1;001.1(r)]TJ.30Td(()Tj
                                                   clear_inverse_video ( slot : IN slot_id )
                                                   delete_char ;
                                                   delete_char_at_left ;
                                                   delete_line ;;I_line        ( text : IN st
                                                               .(c)d(ontent)TgBar020TdIN()Ejnac
                                                               insert_string ( text : IN stri
                                                               locate_slot ( slot : IN slot_i
                                                               move_cs ( old_cs : IN slot_id
```

```
OPERATION_CONTROL_STRUCTURES

  OPERATION clear_error_display
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION ced

  OPERATION clear_inverse_video ( slot : IN slot_id )
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION civ

  OPERATION delete_char
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION dc

  OPERATION delete_char_at_left
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION dcal

  OPERATION delete_line
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION dl

  OPERATION display_cl_line ( text : IN string )
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION dcll

  OPERATION display_content ( content : IN content )
    PSEUDO_CODE
      OPERATION_REQUIREMENTS
        WRITES_TO
          vdu ;
      END_OPERATION_REQUIREMENTS
  END_OPERATION dc
```

```
        OPERATION_REQUIREMENTS
          WRITES_TO
            vdu ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION pclc

    OPERATION reset_dm
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          WRITES_TO
            vdu ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION rdm

    OPERATION ring_bell
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          WRITES_TO
            vdu ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION rb

    OPERATION set_cs ( cs : IN slot_id )
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          WRITES_TO
            vdu ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION scs

    OPERATION set_inverse_video ( slot : IN slot_id )
      PSEUDO_CODE
        OPERATION_REQUIREMENTS
          WRITES_TO
            vdu ;
        END_OPERATION_REQUIREMENTS
    END_OPERATION siv

END_OBJECT dm



OBJECT em IS PASSIVE
  PROVIDED_INTERFACE
    OPERATIONS
      escape_seen ;
      init_em ;
      report_error ( text : IN string ) ;
  REQUIRED_INTERFACE
    OBJECT dm
      OPERATIONS
        clear_error_display ;
        display_error_message ( text : IN string ) ;
  INTERNALS
    OPERATIONS
      add_to_queue ( text : IN string ) ;
      display_error_message ( text : IN string ) ;
      escape_seen ;
      init_em ;
```

```
      remove_from_queue RETURN string ;
      report_error ( text : IN string ) ;
    DATA
      display_in_use : boolean ;
      error_queue  : list_strings ;

    OPERATION_CONTROL_STRUCTURES

      OPERATION report_error ( text : IN string )
        USED_OPERATIONS
          add_to_queue ( text : IN string ) ;
          display_error_message ( text : IN string ) ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            READS_FROM
              display_in_use ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION report_error

      OPERATION escape_seen
        USED_OPERATIONS
          remove_from_queue RETURN string ;
          display_error_message ( text : IN string ) ;
          dm.clear_error_display ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              display_in_use ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION escape_seen

      OPERATION add_to_queue ( text : IN string )
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            REQUIRES_TYPE
              list ;
            WRITES_TO
              error_queue ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION add_to_queue

      OPERATION remove_from_queue RETURN string
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            REQUIRES_TYPE
              list ;
            WRITES_TO
              error_queue ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION remove_from_queue

      OPERATION display_error_message ( text : IN string )
        USED_OPERATIONS
          dm.display_error_message ( text : IN string ) ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            WRITES_TO
              display_in_use ;
```

```
          END_OPERATION_REQUIREMENTS
      END_OPERATION display_error_message

      OPERATION init_em
        USED_OPERATIONS
          dm.clear_error_display ;
        PSEUDO_CODE
          OPERATION_REQUIREMENTS
            REQUIRES_TYPE
              list ;
            WRITES_TO
              display_in_use ;
              error_queue ;
          END_OPERATION_REQUIREMENTS
      END_OPERATION init_em

END_OBJECT em


OBJECT data_types IS PASSIVE
  PROVIDED_INTERFACE
    TYPES
      a_slot ;
      row_position ;
      column_position ;
      content ;
      cursor_position ;
      expression ;
      full_slot_type ;   --{ type+  extended type of a slot
                   false | blank | comment | fp_value | expression }--
      full_value ; --{ value+   extended value of a slot expression
               false | <fp_value> | <s_string> }--
      list_slot_ids ;
      list_strings ;
      operation_math ; --{ + | - | *  | / }--
      operation_full ; --{ <operation_math> | <operation_text> }--
      operation_text ; --{ 'add' | 'subtract' | 'multiply' | 'divide' }--
      s_string ;
      slot_array ;
      slot_id ;
      slot_letter ;
      slot_number ;
      slot_type ; --{ blank | comment | fp_value | expression }--
      slots_index ;
      .0R30.12Tf10.02l1.430Td(|)Tj10.43Td(--)TjmU40.02l1.430.430Td(--)Tj/R410.12Tf10.020Td(f)Tj/R30
```

```
      zero_float   : float ;
END_OBJECT data_types
```

B.3 ***TriviCalc***

```
                                    ->DATA_TYPES.VALIDITY'),
    C('<operation>CLI.COMMAND_DESPATCHER:$STANDARD.STRING
                                    ->DATA_TYPES.VALIDITY'),
    C('<data>CLI.LOAD_IN_PROGRESS'),
  %],
  [%
    C('<constant>DATA_TYPES.ZERO_FLOAT:->$STANDARD.FLOAT'),
    C('<constant>DATA_TYPES.NIL:->POP_11.LIST'),
    C('<constant>DATA_TYPES.MAX_S_STRING:->$STANDARD.INTEGER'),
    C('<constant>DATA_TYPES.MIN_ROW:->$STANDARD.INTEGER'),
    C('<constant>DATA_TYPES.MAX_ROW:->$STANDARD.INTEGER'),
    C('<constant>DATA_TYPES.MIN_LETTER:->$STANDARD.CHARACTER'),
    C('<constant>DATA_TYPES.MAX_LETTER:->$STANDARD.CHARACTER'),
    C('<constant>DATA_TYPES.MIN_FLOAT:->$STANDARD.FLOAT'),
    C('<constant>DATA_TYPES.MAX_FLOAT:->$STANDARD.FLOAT'),
    C('<constant>DATA_TYPES.MIN_COLUMN:->$STANDARD.INTEGER'),
    C('<constant>DATA_TYPES.MAX_COLUMN:->$STANDARD.INTEGER'),
    C('<type>DATA_TYPES.VALUE'),
    C('<type>DATA_TYPES.VALIDITY'),
    C('<type>DATA_TYPES.SLOTS_INDEX'),
    C('<type>DATA_TYPES.SLOT_TYPE'),
    C('<type>DATA_TYPES.SLOT_NUMBER'),
    C('<type>DATA_TYPES.SLOT_LETTER'),
    C('<type>DATA_TYPES.SLOT_ID'),
    C('<type>DATA_TYPES.SLOT_ARRAY'),
    C('<type>DATA_TYPES.S_STRING'),
    C('<type>DATA_TYPES.OPERATION_TEXT'),
    C('<type>DATA_TYPES.OPERATION_FULL'),
    C('<type>DATA_TYPES.OPERATION_MATH'),
    C('<type>DATA_TYPES.LIST_STRINGS'),
    C('<type>DATA_TYPES.LIST_SLOT_IDS'),
    C('<type>DATA_TYPES.FULL_VALUE'),
    C('<type>DATA_TYPES.FULL_SLOT_TYPE'),
    C('<type>DATA_TYPES.EXPRESSION'),
    C('<type>DATA_TYPES.CURSOR_POSITION'),
    C('<type>DATA_TYPES.CONTENT'),
    C('<type>DATA_TYPES.COLUMN_POSITION'),
    C('<type>DATA_TYPES.ROW_POSITION'),
    C('<type>DATA_TYPES.A_SLOT'),
  %],
  [%
    C('<operation>DM.SET_INVERSE_VIDEO:DATA_TYPES.SLOT_ID'),
    C('<operation>DM.SET_CS:DATA_TYPES.SLOT_ID'),
    C('<operation>DM.RING_BELL'),
    C('<operation>DM.RESET_DM'),
    C('<operation>DM.POSITION_CL_CURSOR:DATA_TYPES.CURSOR_POSITION'),
    C('<operation>DM.MOVE_CS:DATA_TYPES.SLOT_ID*DATA_TYPES.SLOT_ID'),
    C('<operation>DM.LOCATE_SLOT:DATA_TYPES.SLOT_ID'),
    C('<operation>DM.INSERT_STRING:$STANDARD.STRING'),
    C('<operation>DM.INSERT_CHAR:$STANDARD.CHARACTER'),
    C('<operation>DM.INIT_DM'),
    C('<operation>DM.DISPLAY_VALUE:DATA_TYPES.SLOT_ID'),
    C('<operation>DM.DISPLAY_ERROR_MESSAGE:$STANDARD.STRING'),
    C('<operation>DM.DISPLAY_CONTENT:DATA_TYPES.CONTENT'),
    C('<operation>DM.DISPLAY_CL_LINE:$STANDARD.STRING'),
    C('<operation>DM.DELETE_LINE'),
    C('<operation>DM.DELETE_CHAR_AT_LEFT'),
    C('<operation>DM.DELETE_CHAR'),
```

```
C('<operation>DM.CLEAR_INVERSE_VIDEO:DATA_TYPES.SLOT_ID'),
C('<operation>DM.CLEAR_ERROR_
```

```
  C('<constant>SA.BLANK:->DATA_TYPES.SLOT_TYPE'),
%],
[%
  C('<operation>WAE.STORE_MACRO:$STANDARD.INTEGER*$STANDARD.STRING'),
  C('<operation>WAE.SET_CS:DATA_TYPES.SLOT_ID'),
  C('<operation>WAE.RECALL_ALL_MACROS:->DATA_TYPES.LIST_STRINGS'),
  C('<operation>WAE.IS_MACRO_NAME:$STANDARD.STRING->$STANDARD.BOOLEAN'),
  C('<operation>WAE.INIT_CS'),
  C('<operation>WAE.INIT_CL'),
  C('<operation>WAE.GET_CS:->DATA_TYPES.SLOT_ID'),
  C('<operation>WAE.EDITOR'),
  [%
    C('<operation>CL.INIT_CL'),<operation>WAEINI
```

```
        C('<operation>CCP.SAVE_CL'),
        C('<operation>CCP.PROCESS_CONTROL_CHAR:$STANDARD.CHARACTER'),
        C('<operation>CCP.IS_CONTROL_CHAR:$STANDARD.CHARACTER
                                  ->$STANDARD.BOOLEAN'),
        C('<constant>CCP.DESPATCH_TABLE:->POP_11.PROPERTY_TABLE'),
      %],
    %],
    [%
      C('<operation>CS.INIT_CS'),
      C('<operation>CS.GET_CS_CONTENT:->DATA_TYPES.CONTENT'),
      C('<operation>CS.SET_CS:DATA_TYPES.SLOT_ID'),
      C('<operation>CS.GET_CS:->DATA_TYPES.SLOT_ID'),
      C('<operation>CS.MOVE_CS_RIGHT'),
      C('<operation>CS.MOVE_CS_LEFT'),
      C('<operation>CS.MOVE_CS_DOWN'),
      C('<operation>CS.MOVE_CS_UP'),
      C('<data>CS.CURRENT_SLOT'),
    %],
    [%
      C('<operation>IL.INIT_IL'),
      C('<operation>IL.IS_MACRO_NAME:$STANDARD.STRING->$STANDARD.BOOLEAN'),
      C('<operation>IL.RECALL_ALL_MACROS:->DATA_TYPES.LIST_STRINGS'),
      C('<operation>IL.RECALL_MACRO:$STANDARD.INTEGER->$STANDARD.STRING'),
      C('<operation>IL.STORE_MACRO:$STANDARD.INTEGER*$STANDARD.STRING'),
      C('<constant>IL.OLD_CL:->$STANDARD.INTEGER'),
      C('<data>IL.MACROS'),
      C('<constant>IL.MIN_MACRO:->$STANDARD.INTEGER'),
      C('<constant>IL.MAX_MACRO:->$STANDARD.INTEGER'),
    %],
  %],
 %],
%]
```

B.3.2 Final *TriviCalc* design Module Structure

Below is the final module structure for the *TriviCalc* problem, proposed by *orp us*.

```
[%
  [%
    [%
      [%
        [%
          [%
            [%
              [%
                C('<constant>DATA_TYPES.MAX_FLOAT:->$STANDARD.FLOAT'),
                C('<constant>DATA_TYPES.MIN_FLOAT:->$STANDARD.FLOAT'),
                C('<operation>SA.IS_FLOAT:$STANDARD.STRING
                                        ->$STANDARD.BOOLEAN'),
              %],
              [%
                C('<constant>DATA_TYPES.MAX_S_STRING:->$STANDARD.INTEGER'),
                C('<operation>SA.IS_COMMENT:$STANDARD.STRING
                                        ->$STANDARD.BOOLEAN'),
              %],
              C('<constant>DATA_TYPES.MAX_LETTER:->$STANDARD.CHARACTER'),
              C('<constant>DATA_TYPES.MAX_ROW:->$STANDARD.INTEGER'),
              C('<constant>DATA_TYPES.MIN_LETTER:->$STANDARD.CHARACTER'),
              C('<constant>DATA_TYPES.MIN_ROW:->$STANDARD.INTEGER'),
              C('<constant>DATA_TYPES.NIL:->POP_11.LIST'),
              C('<constant>DATA_TYPES.ZERO_FLOAT:->$STANDARD.FLOAT'),
              C('<constant>SA.BLANK:->DATA_TYPES.SLOT_TYPE'),
              C('<constant>SA.COMMENT:->DATA_TYPES.SLOT_TYPE'),
              C('<constant>SA.EXPRESSION:->DATA_TYPES.SLOT_TYPE'),
              C('<constant>SA.FLOAT:->DATA_TYPES.SLOT_TYPE'),
              C('<operation>SA.CREATE_SA'),
              C('<operation>SA.CREATE_SLOT:DATA_TYPES.COLUMN_POSITION*
                                         DATA_TYPES.ROW_POSITION
                                        ->DATA_TYPES.A_SLOT'),
              C('<operation>SA.DISPLAY_VALUE:DATA_TYPES.SLOT_ID'),
              C('<operation>SA.EVALUATE:DATA_TYPES.SLOT_ID
                                        ->DATA_TYPES.FULL_VALUE'),
              C('<operation>SA.INIT_SA'),
              C('<operation>SA.IS_SLOT_ARITHMETIC:DATA_TYPES.SLOT_ID
                                        ->$STANDARD.BOOLEAN'),
              C('<type>DATA_TYPES.COLUMN_POSITION'),
              C('<type>DATA_TYPES.CONTENT'),
              C('<type>DATA_TYPES.FULL_VALUE'),
              C('<type>DATA_TYPES.OPERATION_MATH'),
              C('<type>DATA_TYPES.ROW_POSITION'),
              C('<type>DATA_TYPES.SLOT_ID'),
              C('<type>DATA_TYPES.SLOT_TYPE'),
            %],
            [%
              C('<data>SA.SLOTS'),
              C('<data>SA.STACK'),          --
              C('<operation>SA.ADDRE1.4cmB(:DO)1000.(AT)1001.3TA
```

TYPES.SLO(S)TjETQq32.400-4.4022.21-122.cmBI/IMtr

C('<operation>SA.ADS

```
                                $STANDARD.BOOLEAN'),
    C('<operation>SA.DEPTH_FIRST_SEARCH:DATA_TYPES.SLOT_ID
                              ->DATA_TYPES.SLOT_ID'),
    C('<operation>SA.GET_CONTENTS:DATA_TYPES.SLOT_ID
                              ->DATA_TYPES.CONTENT'),
    C('<operation>SA.GET_VALUE:DATA_TYPES.SLOT_ID*$STANDARD.BOOLEAN
                              ->DATA_TYPES.VALUE'),
    C('<operation>SA.IS_SLOT_BLANK:DATA_TYPES.SLOT_ID
                              ->$STANDARD.BOOLEAN'),
    C('<operation>SA.IS_SLOT_COMMENT:DATA_TYPES.SLOT_ID
                              ->$STANDARD.BOOLEAN'),
    C('<operation>SA.IS_SLOT_EXPRESSION:DATA_TYPES.SLOT_ID
                              ->$STANDARD.BOOLEAN'),
    C('<operation>SA.IS_SLOT_FLOAT:DATA_TYPES.SLOT_ID
                              ->$STANDARD.BOOLEAN'),
    C('<operation>SA.LIST_SUCCESSORS:DATA_TYPES.SLOT_ID
                              ->DATA_TYPES.LIST_SLOT_IDS'),
    C('<operation>SA.REMOVE_SUCCESSOR:DATA_TYPES.SLOT_ID*
                                DATA_TYPES.SLOT_ID'),
    C('<operation>SA.UPDATE_ORDER:DATA_TYPES.SLOT_ID
                              ->DATA_TYPES.LIST_SLOT_IDS'),
    C('<type>DATA_TYPES.A_SLOT'),
    C('<type>DATA_TYPES.LIST_SLOT_IDS'),
    C('<type>DATA_TYPES.SLOTS_INDEX'),
    C('<type>DATA_TYPES.SLOT_ARRAY'),
    C('<type>DATA_TYPES.VALUE'),
  %],
  C('<operation>SA.IS_SUCCESSOR:DATA_TYPES.SLOT_ID*
                                DATA_TYPES.SLOT_ID
                              ->$STANDARD.BOOLEAN'),
%],
C('<operation>SA.SAVE_SA:->DATA_TYPES.LIST_STRINGS'),
C('<operation>SA.SET
```

```
      C('<operation>CLI.STORE_COMMENT:DATA_TYPES.LIST_STRINGS
                                  ->DATA_TYPES.VALIDITY'),
      C('<operation>SA.IS_SLOT:$STANDARD.STRING->$STANDARD.BOOLEAN'),
      C('<type>DATA_TYPES.EXPRESSION'),
      C('<type>DATA_TYPES.FULL_SLOT_TYPE'),
      C('<type>DATA_TYPES.SLOT_LETTER'),
      C('<type>DATA_TYPES.SLOT_NUMBER'),
      C('<type>DATA_TYPES.S_STRING'),
    %],
    C('<data>CLI.LOAD_IN_PROGRESS'),
    C('<operation>CLI.COMMAND_DESPATCHER:$STANDARD.STRING
                                  ->DATA_TYPES.VALIDITY'),
    C('<operation>CLI.FULL_FILE_NAME:$STANDARD.STRING->$STANDARD.STRING'),
    C('<operation>CLI.IS_FILE_NAME:$STANDARD.STRING->$STANDARD.BOOLEAN'),
    C('<operation>CLI.LOAD_FILE:DATA_TYPES.LIST_STRINGS
                                  ->DATA_TYPES.VALIDITY'),
    C('<operation>CLI.STORE_VALUE:DATA_TYPES.LIST_STRINGS
                                  ->DATA_TYPES.VALIDITY'),
  %],
%],
[%
  C('<constant>CCP.DESPATCH_TABLE:->POP_11.PROPERTY_TABLE'),
  C('<operation>CCP.CCP_CLI'),
  C('<operation>CCP.CCP_CLI_KEEP'),
  C('<operation>CCP.CCP_DELETE_CHAR'),
  C('<operation>CCP.CCP_DELETE_CHAR_LEFT'),
  C('<operation>CCP.CCP_DELETE_LINE'),
  C('<operation>CCP.CCP_GET_CS'),
  C('<operation>CCP.CCP_GET_CS_CONTENT'),
  C('<operation>CCP.CCP_REPLACE_PERCENT'),
  C('<operation>CCP.OBEY_CL:->DATA_TYPES.VALIDITY'),
  C('<operation>CCP.PROCESS_CONTROL_CHAR:$STANDARD.CHARACTER'),
  C('<operation>CCP.SAVE_CL'),
%],
[%
  C('<constant>CL.MAX_CURSOR:->$STANDARD.INTEGER'),
  C('<constant>CL.MAX_LENGTH:->$STANDARD.INTEGER'),
  C('<constant>CL.MIN_CURSOR:->$STANDARD.INTEGER'),
  C('<constant>CL.MIN_LENGTH:->$STANDARD.INTEGER'),
  C('<data>CL.CURSOR'),
  C('<data>CL.EOS'),
  C('<data>CL.LINE'),
  C('<operation>CCP.REPLACE_CL:$STANDARD.STRING'),
  C('<operation>CL.CURSOR_LEFT'),
  C('<operation>CL.CURSOR_RIGHT'),
  C('<operation>CL.DELETE_CHAR'),
  C('<operation>CL.DELETE_CHAR_LEFT'),
  C('<operation>CL.DELETE_LINE'),
  C('<operation>CL.GET_CL:->$STANDARD.STRING'),
  C('<operation>CL.INIT_CL'),
  C('<operation>CL.INSERT_CHAR:$STANDARD.CHARACTER'),
  C('<operation>CL.INSERT_STRING:$STANDARD.STRING'),
  C('<operation>CL.LOCATE_SOL'),
  C('<operation>CL.REPLACE_PERCENT:->DATA_TYPES.VALIDITY'),
  C('<operation>DM.POSITION_CL_CURSOR:DATA_TYPES.CURSOR_POSITION'),
  C('<operation>WAE.INIT_CL'),
  C('<type>DATA_TYPES.CURSOR_POSITION'),
%],
```

```
[%
  C('<constant>IL.MAX_MACRO:->$STANDARD.INTEGER'),
  C('<constant>IL.MIN_MACRO:->$STANDARD.INTEGER'),
  C('<constant>IL.OLD_CL:->$STANDARD.INTEGER'),
  C('<data>IL.MACROS'),
  C('<operation>CCP.CCP_RECALL_MACRO'),
  C('<operation>CCP.CCP_STORE_MACRO'),
  C('<operation>CLI.STORE_MACRO:DATA_TYPES.LIST_STRINGS
                                   ->DATA_TYPES.VALIDITY'),
  C('<operation>IL.INIT_IL'),
  C('<operation>IL.IS_MACRO_NAME:$STANDARD.STRING->$STANDARD.BOOLEAN'),
  C('<operation>IL.RECALL_ALL_MACROS:->DATA_TYPES.LIST_STRINGS'),
  C('<operation>IL.RECALL_MACRO:$STANDARD.INTEGER->$STANDARD.STRING'),
  C('<operation>IL.STORE_MACRO:$STANDARD.INTEGER*$STANDARD.STRING'),
  C('<operation>WAE.IS_MACRO_NAME:$STANDARD.STRING->$STANDARD.BOOLEAN'),
  C('<operation>WAE.STORE_MACRO:$STANDARD.INTEGER*$STANDARD.STRING'),
%],
[%
  C('<data>CS.CURRENT_SLOT'),
  C('<operation>CCP.CCP_ADJUST'),
  C('<operation>CS.GET_CS:->DATA_TYPES.SLOT_ID'),
  C('<operation>CS.GET_CS_CONTENT:->DATA_TYPES.CONTENT'),
  C('<operation>CS.INIT_CS'),
  C('<operation>CS.MOVE_CS_DOWN'),
  C('<operation>CS.MOVE_CS_LEFT'),
  C('<operation>CS.MOVE_CS_RIGHT'),
  C('<operation>CS.MOVE_CS_UP'),
  C('<operation>CS.SET_CS:DATA_TYPES.SLOT_ID'),
  C('<operation>DM.MOVE_CS:DATA_TYPES.SLOT_ID*DATA_TYPES.SLOT_ID'),
  C('<operation>WAE.GET_
```

```
      C('<operation>EM.DISPLAY_ERROR_MESSAGE:$STANDARD.STRING'),
      C('<operation>EM.ESCAPE_SEEN'),
      C('<operation>EM.INIT_EM'),
      C('<operation>EM.REMOVE_FROM_QUEUE:->$STANDARD.STRING'),
      C('<operation>EM.REPORT_ERROR:$STANDARD.STRING'),
    %],
  %],
  [%
    [%
      C('<data>EDITOR.CHANNEL'),
      C('<operation>EDITOR.GET_CHAR:->$STANDARD.CHARACTER'),
      C('<operation>EDITOR.INIT_CP'),
      C('<operation>EDITOR.TERM_CP'),
    %],
    C('<operation>CCP.IS_CONTROL_CHAR:$STANDARD.CHARACTER->$STANDARD.BOOLEAN'),
    C('<operation>CP.EDITOR'),
    C('<operation>EDITOR.EDITOR'),
    C('<operation>EDITOR.IS_ESCAPE:$STANDARD.CHARACTER->$STANDARD.BOOLEAN'),
    C('<operation>EDITOR.IS_PRINTABLE:$STANDARD.CHARACTER->$STANDARD.BOOLEAN'),
    C('<operation>WAE.EDITOR'),
  %],
  C('<operation>CLI.MAIN_PROGRAM'),
  C('<operation>CLI.REINITIALISE_SYSTEM'),
  C('<operation>CLI.SAVE_FILE:DATA_TYPES.LIST_STRINGS->DATA_TYPES.VALIDITY'),
  C('<operation>TRIVICALC.MAIN_PROGRAM'),
  C('<operation>WAE.RECALL_ALL_MACROS:->DATA_TYPES.LIST_STRINGS'),
%],
[%
  C('<constant>$STANDARD.FALSE:->$STANDARD.BOOLEAN'),
  C('<operation>$STANDARD.<<=>>:$STANDARD.STRING*$STANDARD.STRING
                                    ->$STANDARD.BOOLEAN'),
  C('<type>$STANDARD.BOOLEAN'),
  C('<type>$STANDARD.CHARACTER'),
  C('<type>$STANDARD.FLOAT'),
  C('<type>$STANDARD.INTEGER'),
  C('<type>$STANDARD.STRING'),
%],
[%
  C('<operation>POP_11.CLOSE_FILE:POP_11.CHANNEL'),
  C('<operation>POP_11.GET_INPUT_CHAR:POP_11.CHANNEL->$STANDARD.CHARACTER'),
  C('<operation>POP_11.ISSTRING:$STANDARD.STRING->$STANDARD.BOOLEAN'),
  C('<operation>POP_11.MATCHES:POP_11.LIST*POP_11.LIST->$STANDARD.BOOLEAN'),
  C('<operation>POP_11.OPEN:$STANDARD.STRING*$STANDARD.STRING
                                    ->POP_11.CHANNEL'),
  C('<operation>POP_11.PARSE_STRING:$STANDARD.STRING->POP_11.LIST'),
  C('<operation>POP_11.READ_LINE:POP_11.CHANNEL->$STANDARD.STRING'),
  C('<operation>POP_11.SYSEXIT'),
  C('<operation>POP_11.WRITE_LINE:POP_11.CHANNEL*$STANDARD.STRING'),
  C('<type>POP_11.CHANNEL'),
  C('<type>POP_11.LIST'),
  C('<type>POP_11.PROPERTY_
```

Appendix C

Glossary and Abbreviations

Abstract Data Types	An object which encapsulates a type and its operations and operators, but without declaring a data item for the type.

Coupling	The dependency between two objects. High coupling indicates that a change to one object is likely to impact on another. Low cohesion is desirable.
Degree	The number of edges incident on a graph's node.
Digraphs	A graph, where the direction of the edges is significant. Also called directed graphs. All graphs in this thesis are digraphs.
DFD	Data Flow Diagram.
Encapsulation	The method of combining data and operations on those data in an object. (Robinson, 1992a, p.228)
Entity (design)	A design component, including objects. In HOOD these consist of objects, types, operations, constants, variables, operation_sets, and exception.
Environmental Object	An object which represents the provided interface of another object used by the system to be designed, but which is not part of the [cur-

Inter-	Prefix meaning among, between, together, one with another, etc. (Bancroft, 1969, p.181)
Interface	Specification of the usable (visible) part of an object.
Internals (	

Overloading	The ability for an operation name to be repeated with the definition of a single object, providing that there is some way of differentiating between them, i.e., by having different parameter and result type profiles in Ada, or different argument signature in C++. (Robinson, 1992a, p.229)
Polymorphism	The ability for the selection of an operation body to be determined at run time, according to the class of the object to which the operation is currently referring. (Robinson, 1992a, p.229)
Provided Interface	Defines the services that an object provides to its clients.
Ratio Scales	A scale with a total ordering permitting statements such as "***A*** is twice as big as ***B***" to be meaningful.
Requires	An edge in a design graph represents a requires relationship, i.e., an

Appendix D

Notation Summary

$\varnothing$	The empty set.
$\boldsymbol{a} \in \boldsymbol{A}$	$\boldsymbol{a}$ is a member of the set $\boldsymbol{A}$.
$\boldsymbol{A} = \boldsymbol{B}$	Set equality.
$\lvert\boldsymbol{A}\rvert$	The cardinality of the set $\boldsymbol{A}$.
$\boldsymbol{A} \cup \boldsymbol{B}$	Set union.
$\boldsymbol{A} \cap \boldsymbol{B}$	Set intersection.
$\boldsymbol{A} \subseteq \boldsymbol{B}$	Subset.
$\boldsymbol{A} \subset \boldsymbol{B}$	Proper subset.
	The finite set of nodes of a graph $\boldsymbol{G}(\quad , \mathcal{E})$.
$\mathcal{E}$	The finite set of edges of a graph $\boldsymbol{G}(\quad , \mathcal{E})$.
$\mathcal{E}^*$	

$\mathrm{KC}(\boldsymbol{x})$	The Kolmogorov Complexity of $\boldsymbol{x}$.
$\boldsymbol{A} \cong \boldsymbol{B}$	Indicates an encoding, such that, $\boldsymbol{B}$ is an encoding of $\boldsymbol{A}$.
$\mathcal{L}^*(\boldsymbol{n})$	An optimal universal prefix code for all positive integers. Each integer has an encoding of the form, $\log_2 \boldsymbol{n} + \log_2 \log_2 \boldsymbol{n} + \log_2 \log_2 \log_2 \boldsymbol{n} + \cdots$, terminating when $\log(\ldots \log \boldsymbol{n}) \leq 0$. See Section 5.2.3.
$\log_2^*(\boldsymbol{n})$	Length (in bits) of the $\mathcal{L}^*(\boldsymbol{n})$ function, given by $\mathcal{L}^*(\boldsymbol{n}) + \log_2 2.865064$. See Section 5.2.3.
	Set of all possible design graphs.
$\Psi(\boldsymbol{G})$	The complexity of the design graph $\boldsymbol{G}$. See Section 6.2.